PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-160856
(43)Date of publication of application : 20.06.1997

(51)Int.Cl. G06F 13/00

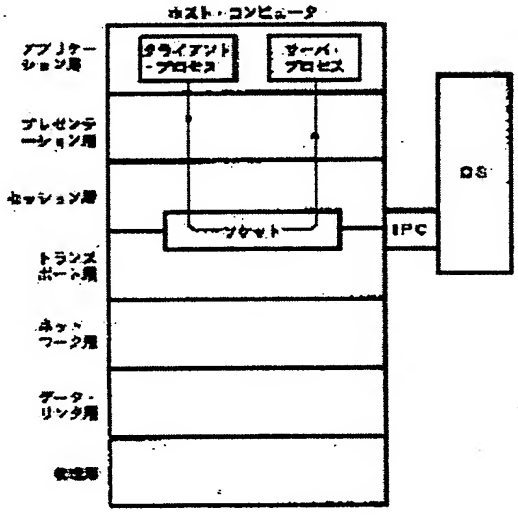
(21)Application number : 08-192028
(22)Date of filing : 22.07.1996

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor : KUMAR S VERDON
SANDAYA CAPUL
I SHU UEI

(30)Priority
Priority number 95 526833 Priority date 12.09.1995 Priority country US

(54) COMMUNICATION MANAGEMENT METHOD AND COMPUTER SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide the method which manages the communication path between a client process and a server process in decentralized calculation environment by residing on a host computer connected to a physical network having a transport layer and a network layer.
SOLUTION: When the client process makes a remote procedure call(RPC), this method is started by detecting whether or not there is the server process discriminated by the RPC. When the process is present, a binding handle is returned to the client process. The protocol order in the binding handle is mapped in 2nd protocol order setting the inter-process communication path between the client and server processes instead of a communication path passing through the transport and network layers of the physical network. Then the RPC is executed by using the function of an operating system.



LEGAL STATUS

CW

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平9－160856

(43)公開日　平成９年(1997)６月20日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　13/00	３５７		Ｇ０６Ｆ　13/00	３５７Ｚ	

審査請求　未請求　請求項の数17　ＯＬ　（全 29 頁）

(21)出願番号　特願平8－192028

(22)出願日　平成８年(1996)７月22日

(31)優先権主張番号　０８／５２６８３３

(32)優先日　1995年９月12日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　クマール・エス・バーダン
アメリカ合衆国　78759、テキサス州、オースチン、ジョリービル・ロード、アパートメント・ナンバー903　11160

(74)代理人　弁理士　合田　潔　（外２名）

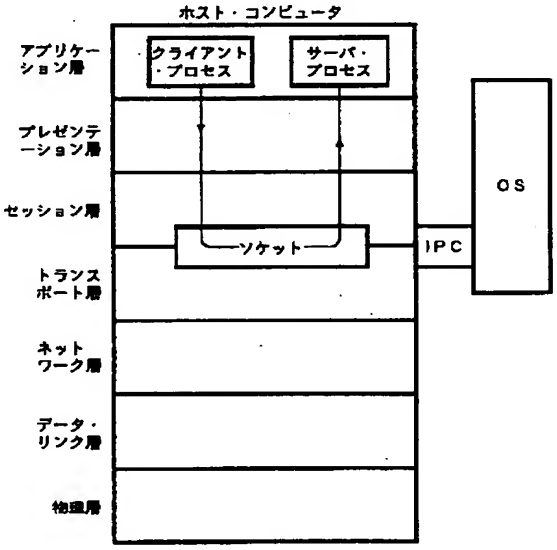
最終頁に続く

(54)【発明の名称】　通信管理方法及びコンピュータ・システム

(57)【要約】

【課題】　トランスポート層及びネットワーク層を持つ物理的ネットワークに接続されたホスト・コンピュータ中に常駐し、分散計算環境内にあるクライアント・プロセス及びサーバ・プロセス間の通信路を管理する方法を与える。

【解決手段】　クライアント・プロセスが遠隔プロシージャ呼び出し（ＲＰＣ）を行なった時、ＲＰＣによって識別されたサーバ・プロセスがホスト・コンピュータ中にあるか否かを検出することにより本発明の方法が開始する。若し存在すれば、バインディング・ハンドルがクライアント・プロセスに返送される。バインディング・ハンドル中のプロトコル順序は、物理的ネットワークのトランスポート及びネットワーク層を通る通信路の代わりに、クライアント及びサーバ・プロセス間のインタープロセス通信路を設定する第２プロトコル順序にマップされる。次に、オペレーティング・システムの機能を用いてＲＰＣが実行される。



1

【特許請求の範囲】

【請求項１】　分散計算環境内におけるクライアント・プロセスとサーバ・プロセスの間の通信を管理する方法であって、上記クライアント・プロセスは、トランスポート層及びネットワーク層を有する物理ネットワークに接続されたホスト・コンピュータ中に存在し、上記方法は、

（ａ）　クライアント・プロセスによって行なわれた遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中に存在するか否かを決定するステップを含み、上記ＲＰＣは、トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を持ち、

（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中に存在するならば、上記ＲＰＣのプロトコル順序には関係なく、上記クライアント・プロセスと上記サーバ・プロセスとの間にインタープロセス通信路を設定するステップと、

（ｃ）　上記ＲＰＣのプロトコル順序を上記クライアント・プロセスに戻すステップと、

（ｄ）　上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、

を含む通信管理方法。

【請求項２】　上記プロトコル順序はＲＰＣに影響を与えないで使用されることを特徴とする請求項１に記載の通信管理方法。

【請求項３】　上記プロトコル順序は接続指向プロトコル順序であることを特徴とする請求項１に記載の通信管理方法。

【請求項４】　上記プロトコル順序は非接続プロトコル順序であることを特徴とする請求項１に記載の通信の管理方法。

【請求項５】　上記遠隔プロシージャ呼び出しは、上記ホスト・コンピュータのオペレーティング・システムのメッセージ送受信機能を用いて実行されることを特徴とする請求項１に記載の通信管理方法。

【請求項６】　分散計算環境内におけるクライアント・プロセスとサーバ・プロセスの間の通信を管理する方法であって、上記クライアント・プロセスは、トランスポート層及びネットワーク層を有する物理ネットワークに接続されたホスト・コンピュータ中に存在し、上記方法は、

（ａ）　遠隔プロシージャ呼び出し（ＲＰＣ）が上記クライアント・プロセスによって行なわれた時、上記遠隔プロシージャ呼び出しによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中に存在しているか否かを決定するステップと、

（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中にあれば、トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を含む第１データ構造を上記クライアント・プロセスに戻すステップと、

2

（ｃ）　上記第１データ構造を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を定義するプロトコル順序を含む第２データ構造にマップするステップと、

（ｄ）　上記第２データ構造中のプロトコル順序によって定義された上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、

を含む通信管理方法。

【請求項７】　上記遠隔プロシージャ呼び出しは、上記ホスト・コンピュータのオペレーティング・システムのメッセージ送受信機能を用いて実行されることを特徴とする請求項６に記載の通信管理方法。

【請求項８】　上記第１データ構造のプロトコル順序は上記ＲＰＣに影響を与えることなく使用されることを特徴とする請求項６に記載の通信管理方法。

【請求項９】　上記第１データ構造のプロトコル順序は接続指向プロトコル順序であることを特徴とする請求項６に記載の通信管理方法。

【請求項１０】　上記第２データ構造のプロトコル順序はソケット・ファイルに対するフル・パス名を含んでいることを特徴とする請求項９に記載の通信管理方法。

【請求項１１】　命名規則及びエンドポイントが上記パス名を決定するのに用いられることを特徴とする請求項１０に記載の通信管理方法。

【請求項１２】　ホスト・コンピュータがＵＮＩＸベースのオペレーティング・システムをサポートしており、クライアント・プロセスがトランスポート層及びネットワーク層を有する物理ネットワークに接続された上記ホスト・コンピュータ中に常駐している分散計算環境内において、上記クライアント・プロセスが遠隔プロシージャ呼び出しを行なった時に、上記クライアント・プロセス及びサーバ・プロセス間の通信を管理する方法であって、

（ａ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中に存在するならば、上記遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を持つバインディング・ハンドルを上記クライアント・プロセスに戻すステップと、

（ｂ）　上記遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を設定する代替プロトコル順序にマップするステップと、

（ｃ）　上記ＵＮＩＸベースのオペレーティング・システムのメッセージ送受信機能を用いて、上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、

を含む通信管理方法。

3

【請求項１３】　「ncacn」は接続指向ＲＰＣプロトコルであり、「unix」は「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）通信ドメイン」を識別し、「stream」はＵＮＩＸドメイン・ソケットを識別するものとした場合、上記遠隔プロシージャ呼び出しに関連したプロトコル順序は接続指向プロトコル「ncacn_ip_tcp」であり、上記代替プロトコル順序は「ncacn_unix_stream」であることを特徴とする請求項１２に記載の通信管理方法。

【請求項１４】　「ncadg」は非接続ＲＰＣプロトコルであり、「unix」は「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）通信ドメイン」を識別し、「dgram」はＵＮＩＸドメイン・ソケットを識別するものとした場合、上記遠隔プロシージャ呼び出しに関連したプロトコル順序は非接続指向プロトコル「ncacn_ip_udp」であり、上記代替プロトコル順序は「ncadg_unix_dgram」であることを特徴とする請求項１２に記載の通信管理方法。

【請求項１５】　ホスト・コンピュータがインタープロセス通信（ＩＰＣ）機構を有し、かつ、ユーザが分散されたリソース及びプロセス・アプリケーションにアクセスすることのできる分散計算環境を与えるローカル・エリア・ネットワークであって、

クライアント・プロセスからの遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出する検出手段と、

該検出手段に応答して、予期されたプロトコル順序を上記クライアント・プロセスに戻し、かつ、上記ＲＰＣを助長するために代替プロトコル順序を用いる手段を含み、上記代替プロトコル順序はＩＰＣ構造を通る通信路を設定することと、

を含むローカル・エリア・ネットワーク。

【請求項１６】　ユーザが分散されたリソース及びプロセス・アプリケーションにアクセスすることのできる分散計算環境を与え、かつ、トランスポート層及びネットワーク層を有するローカル・エリア・ネットワークに接続されたホスト・コンピュータを含むコンピュータ・システムであって、

クライアント・プロセスからの遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出する検出手段と、

該検出手段に応答して、予期されたプロトコル順序を上記クライアント・プロセスに戻し、かつ、上記ＲＰＣを助長するために代替プロトコル順序を用いる手段を含み、上記代替プロトコル順序の使用は上記クライアント・プロセスに影響を与えないことと、

を含むコンピュータ・システム。

【請求項１７】　ホスト・コンピュータによって読み取り可能であり、かつ、上記ホスト・コンピュータで実行されるクライアント・プロセスからの通信を管理する方法を実施するための、該ホスト・コンピュータによって実行可能な命令からなるプログラムを具現化するプログラム記憶装置であって、上記ホスト・コンピュータはトランスポート層及びネットワーク層を有するローカル・エリア・ネットワークに接続されており、上記方法は、

（ａ）　遠隔プロシージャ呼び出しが上記クライアント・プロセスによって行なわれた時、上記遠隔プロシージャ呼び出しによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出するステップと、

（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中にあるならば、上記トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を含む第１データ構造を上記クライアント・プロセスに戻すステップと、

（ｃ）　上記第１データ構造を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を定義するプロトコル順序を含む第２データ構造にマップするステップと、

（ｄ）　上記第２データ構造のプロトコル順序によって定義された上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、を含むプログラム記憶装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、ネットワーク通信、より詳細に言えば、コンピュータ・ネットワーク中のクライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータにおいてサポートされている時、クライアント・プロセス及びサーバ・プロセスとの間の遠隔プロシージャ呼び出しを効率的に管理する方法に関する。

【０００２】

【従来の技術】複数のコンピュータが、情報を交換し、かつリソースを共有することができるように、ローカル・エリア・ネットワーク（ＬＡＮ）に複数のコンピュータを相互に接続する技術は公知である。ローカル・エリア・ネットワークは、分散されたリソースと複数のコンピュータ中のプロセス・アプリケーションとに対して、ユーザがアクセスできるような分散計算環境を与える。ネットワーク通信は、いわゆる通信プロトコルを用いて遂行される。この通信規則によって、ローカル・エリア・ネットワークにおける通信アーキテクチャは、通常、物理層、論理リンク層、ネットワーク層、トランスポート層、セッション層、プレゼンテーション層及びアプリケーション層からなる７層モデルに標準化したものとして特徴付けられる。物理層は、情報を伝送するために使用される実際の物理的な装置及び媒体を含んでいる。論

理リンク層は、データ・パケットをフレームし、かつ物理層のデータ・フローを制御して、実際の物理的媒体とは無関係にデータのデリバリーを保証する。ネットワーク層は、データ・パケットをアドレスし、そして経路指定を行なう。この層は、ソース・ノードと宛先ノードとの間で、ネットワーク中の経路を作成し、そして維持する。トランスポート層は、ノード間の伝送パイプラインを作成し、そしてネットワーク層との接続を管理する。セッション層は、通常、遠隔プロシージャ呼び出し（ＲＰＣ）のサポートを与え、ノード間の接続の一貫性を維持し、そしてデータ交換を制御する。プレゼンテーション層は、データをエンコードし、かつデコードし、そしてノード間の透過性動作態様を与える。最後に、アプリケーション層は、エンド・ユーザのプロセスに対してインターフェースを与え、かつアプリケーションに対して標準化されたサービスを与える。

【０００３】７層モデルは特定のネットワークに応じて多くの変形を持っている。従って、例えばＡＩＸオペレーティング・システムの下でＩＢＭ社のRISC System/6000（商標）のコンピュータ・ワークステーションを動作するＴＣＰ／ＩＰ（Transmission Control Protocol/Internet Protocol）アーキテクチャに基づいたネットワーク・アーキテクチャにおいては、セッション層とトランスポート層との間に存在するソケット層と呼ばれる他の層がある。ソケット層は、物理的なポートと等価な論理構造を持つ所謂「ソケット」を作成する。このアーキテクチャにおいて、ＲＰＣ機構は、セッション層においてサポートされるばかりでなく、セッション層の機能も含んでいる。分散計算環境（ＤＣＥ）において有用な周知のＲＰＣ機構は「オープン・システムズ・ファンデーション（ＯＳＦ）」によって与えられたソフトウェア・コードを含んでいる。

【０００４】ＯＳＦ－ＤＣＥ－ＲＰＣ機構は、クライアントが遠隔プロシージャ呼び出し（ＲＰＣ）を用いてサーバにサービスを要求する場合に、「クライアント」及び「サーバ」間の通信を管理するために従来から使用されている。「クライアント」とは、計算環境内の何処かでアクセス可能なサービスを要求したネットワークの参加者のことである。「サーバ」は、クライアントにより要求されたサービスを提供する。ＯＳＦ－ＤＣＥ－ＲＰＣ機構において、各クライアント・プロセス（クライアントのコンピュータ中で実行されるプロセス）は、ソケット層によって作成される１つの関連ソケットを持っている。同様に、各サーバ・プロセスは１つのソケットに関連付けられる。ＲＰＣに応答して、呼び出しディレクトリ・サービスは、サーバ・プロセスが実行されるネットワーク・アドレス及びポート番号として、サーバ・プロセスの位置を特定する「バインディング・ハンドル」と呼ばれるデータ構造を戻す。次に、バインディング・ハンドルは、クライアント・プロセスとサーバ・プロセスとの間の通信路を定義するためにＲＰＣ機構によって使用される。この通信路は、ソケットをオープンするために、「インターネット・ネットワーク・アドレス・ファミリィ（ＡＦ＿ＩＮＥＴ）」のｉｐベース（即ち、ネットワーク層）プロトコル順序を用いて定義される。この通信路は、クライアント・プロセスから出発して、トランスポート層及びネットワーク層を通ってネットワークへ出た後に、サーバ・プロセスが実行されるホスト・コンピュータに関連した層へ戻るループ路を構成している。

【０００５】上述のＯＳＦ－ＤＣＥ－ＲＰＣ機構は、クライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータ中で動作されているか否かを判別することはできない。すべての場合において、この機構は、トランスポート（ＴＣＰまたはＵＤＰ）層及びネットワーク（ＩＰ）層を通過する通信路を設定するＡＦ＿ＩＮＥＴプロトコル順序を含むクライアント・プロセスに、バインディング・ハンドルを戻す。ＴＣＰを通る通信は、接続指向プロトコル順序を使用するけれども、ＵＤＰを通る通信は、非接続プロトコル順序を使用する。然しながら、いずれの場合においても、クライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータ中に存在する時には、ＲＰＣは、所謂ループバック・メッセージを生成する。何故ならば、ネットワーク層（ＩＰ層）が宛先ネットワーク・アドレスを受け取ったならば、ＩＰ層は、ＲＰＣが「ローカル」であると認識し、従って、その通信路がトランスポート層を通ってアプリケーション層上のサーバ・プロセスにループ・バックされねばならないと認識するからである。このループ・バックの要件のために、同じコンピュータ中のクライアント・プロセス及びサーバ・プロセス間のＲＰＣは、性能の観点から見て最適化されていない。何故ならば、それらのＲＰＣは、必要としないトランスポート層及びネットワーク層を使用するからである。

【０００６】

【発明が解決しようとする課題】従って、本発明の主目的は、ネットワーク中の同じホスト・コンピュータ中で実行しているクライアント・プロセス及びサーバ・プロセス間の遠隔プロシージャ呼び出し（ＲＰＣ）を効率的に管理することにある。

【０００７】本発明の他の目的は、クライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータ中で実行されているか否かを、ネットワークのＲＰＣ機構によって識別させ、若しクライアント及びサーバ・プロセスの両方が同じホスト・コンピュータ中で実行されているならば、ローカル・インタープロセス通信（ＩＰＣ）機能を活用する代替プロトコル順序を用いてネットワーク層及びトランスポート層をバイパスさせることにある。

【０００８】本発明のより特別の目的は、ＤＣＥ内のク

ライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータ中に存在する場合を検知し、これにより、遠隔プロシージャ呼び出しを動作させるためのネットワーク層（ＩＰ）またはトランスポート層（ＴＰ）の経路の代わりに、ローカルＩＰＣ機構を使用することにある。ＩＰ層またはＴＰ層をバイパスさせることは、性能に顕著な向上を与える。

【０００９】本発明の他の目的は、同じホスト・コンピュータ中で実行されている２つのプロセスの間で行なわれるＲＰＣである「ローカル」ＲＰＣに影響するためのＩＰＣ機構を使用することにある。特定の実施例において、ＩＰＣ機構は「ＵＮＩＸドメイン」ソケットであって、本発明は「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）」に基づいた上述のソケットをオープンするために、接続指向プロトコル順序を使用している。ＡＦ＿ＵＮＩＸを使用することによって、そのオペレーティング・システム・カーネルは、同じホスト・コンピュータ中の２つのプロセス間のブリッジ通信のタスクを操作する。

【００１０】本発明の他の目的は、サーバ・プロセス及びクライアント・プロセスが同じホスト・コンピュータ中で動作している時に、ローカルＲＰＣ機構の使用を自動化することにある。サーバ・プロセス及びクライアント・プロセスが同じホスト・コンピュータ中に存在する時に、アプリケーションがＲＰＣのローカル性に気づくことなく、ローカルＲＰＣ用のプロトコル順序が使用される。

【００１１】

【課題を解決するための手段】ローカルＲＰＣ機能は常に使用可能にされていることが望ましく、そして「ローカル」ＲＰＣはクライアント・プロセスに対して透過的な動作態様を持っている。従って、ローカル・サーバ・プロセスに対するＲＰＣの検出に応答して、予期されたプロトコル順序が、クライアント・プロセスに戻され、他方、ホスト・コンピュータのＩＰＣ機構を通るＲＰＣを促進するために、代替のプロトコル順序が使用される。

【００１２】本発明の上述の目的及び他の目的は、トランスポート層及びネットワーク層を有する物理的ネットワークに接続されたホスト・コンピュータ中に常駐するクライアント・プロセス及びサーバ・プロセス間の通信であり、かつ分散計算環境内にある通信を管理するための方法によって具現化される。本発明の方法は、クライアント・プロセスが遠隔プロシージャ呼び出し（ＲＰＣ）を行なった時に、遠隔プロシージャ呼び出しによって識別されたサーバ・プロセスが同じホスト・コンピュータ中に存在するか否かを検出することによって開始する。若し、クライアント・プロセスとサーバ・プロセスとが同じホスト・コンピュータ中に存在すれば、物理的ネットワークのトランスポート層及びネットワーク層を通る通信路の代わりに、クライアント・プロセス及びサーバ・プロセス間にインタープロセス通信路を設定するプロトコル順序を持つ少なくとも１つのバインディング・ハンドルを含んでいるバインディング・ハンドル・ベクトルがクライアント・プロセスに戻される。次に、望ましくはホスト・コンピュータのオペレーティング・システムの送信及び受信メッセージ伝達機能を用いて、遠隔プロシージャ呼び出しが実行される。

【００１３】本発明の良好な実施例に従った「ローカル」ＲＰＣは、クライアント・プロセスに対する影響が透過的である（クライアント・プロセスに対して影響を与えない）。従って、ローカル・サーバ・プロセスに対するＲＰＣの検出に応答して、予期されたプロトコル順序がクライアント・プロセスに戻され、他方、ＲＰＣを助長するために、代替プロトコル順序が使用される。代替プロトコル順序は同じホスト・コンピュータのＩＰＣ機構を通る通信路を設定する。

【００１４】より包括的に言えば、本発明の方法は、トランスポート層及びネットワーク層の両方を使用する通信路を定義するプロトコル順序を含む第１のデータ構造をクライアント・プロセスへ先ず戻すことによって、ローカルＲＰＣに応答する。次に、第１のデータ構造は、クライアント・プロセス及びサーバ・プロセス間のインタープロセス通信路を定義するプロトコル順序を含む第２のデータ構造にマップされる。次に、遠隔プロシージャ呼び出しは、第２のデータ構造中のプロトコル順序によって定義されたインタープロセス通信路を介して実行される。従って、本発明の良好な実施例において、第１のデータ構造のプロトコル順序は、ＲＰＣに影響を与えないで使用される。

【００１５】ホスト・コンピュータがＵＮＩＸベースのオペレーティング・システムをサポートしている特定の実施例において、本発明の方法は、ＲＰＣによって識別されたサーバ・プロセスが同じホスト・コンピュータ中に存在するか否かを検出することによってＲＰＣに応答する。若し、ＲＰＣによって定義されたサーバ・プロセスが同じホスト・コンピュータ中に存在しているならば、本発明の方法は、遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を持っているバインディング・ハンドルをクライアント・プロセスに戻す。遠隔プロシージャ呼び出しに関連しているプロトコル順序は、通常、接続指向プロトコル「ncacn_ip_tcp」か、または非接続プロトコル「ncadg_ip_udp」かの何れかである。次に、プロトコル順序は、クライアント・プロセス及びサーバ・プロセス間のインタープロセス通信路を設定する代替プロトコル順序にマップされる。ＲＰＣが接続指向プロトコルを識別した場合には、このステップは、「ncacn_ip_tcp」をプロトコル順序「ncacn_unix_stream」にマップする。ＲＰＣが非接続プロトコルを識別した場合には、このステップは、「ncadg_ip_udp」を「nc

adq_unix_dgram」プロトコル順序にマップする。次に、クライアント・プロセスは、クライアント・プロセスが予期したプロトコル順序を「検出する（見る）」けれども、しかし、そのプロトコル順序は、ＲＰＣに影響せずに使用される。上述の動作態様とは異なって、本発明の方法は、代表例において、ＵＮＩＸベースのオペレーティング・システムのメッセージ送受信機能を用いることによって、クライアント・プロセスに対して透過的に実行される。

【００１６】上述の記載は、本発明に特に関連を持つ幾つかの目的の概略を説明したものである。これらの目的は、本発明の幾つかの特別な特徴及び適用例について単に説明したものとして理解されるべきである。本発明の実施の態様は以下に説明する通りであり、本発明を適用することによって、多くの他の有益な結果を得ることができる。

【００１７】

【発明の実施の形態】上述したように、本発明は、総括的に言えば、複数のコンピュータに分散されたリソース及びプロセス・アプリケーションに、ユーザがアクセスすることのできる分散計算環境を提供するローカル・エリア・ネットワーク中のクライアント・プロセス及びサーバ・プロセス間の通信を管理することに向けられている。従来の分散計算環境（ＤＣＥ）は、図１に示されており、そしてネットワーク１４を介してサーバ１２に相互接続されたクライアント１０を含んでいる。クライアント１０及びサーバ１２の各々はコンピュータである。例えば、各コンピュータは、ＡＩＸ（Advanced Interactive Executive）オペレーティング・システムを実行するＩＢＭ（商標）社のRISC System/6000（商標）（縮小されたインストラクション・セット、即ち、所謂ＲＩＳＣをベースとしたワークステーション）であってもよい。ＡＩＸオペレーティング・システムは、アプリケーションのインターフェース・レベルにおいて、ＡＴ＆Ｔ社のＵＮＩＸオペレーティング・システムのバージョン５．２と互換性を持っている。ＲＩＳＣベースのワークステーション型コンピュータの種々のモデルは、例えば刊行物「RISC System/6000, 7073 and 7016 POWERstation and POWERserver Hardware Technical Reference, -- Order No. SA23-2644-00」など、ＩＢＭ社の多くの刊行物に記載されている。ＡＩＸオペレーティング・システムは、ＩＢＭ社で刊行された刊行物「AIX Operating System Technical Reference, First Edition（１９８５年１１月）及び他の刊行物に記載されている。ＵＮＩＸオペレーティング・システムの設計に関する細部については、Prentice-Hall社により刊行された（１９８６年）バッハ（Maurice J. Bach）著の刊行物「Design of the Unix Operating System」を参照されたい。

【００１８】本発明は、特定の１実施例において、ＩＢＭのシステム・ネットワーク・アーキテクチャ（ＳＮＡ）、より特定して言えばＳＮＡのＬＵ６．２の拡張プログラム間通信（Advanced Program to Program Communication-ＡＰＰＣ）によって相互接続された複数個のＩＢＭ社のRISC System/6000において実行される。ＳＮＡは、そのリンク・レベルとして、ゼロックス社によって開発されたローカル・エリア・ネットワーク（ＬＡＮ）のイーサネットまたはＳＤＬＣ（同期データ・リンク制御）を使用する。ローカル・エリア・ネットワークの簡単な説明は、ブレディ（Robert J. Brady、Prentice-Hall社）により刊行された（１９８３年）ジョーダン及びチャーチル（Larry E. Jordan and Bruce Churchill）共著の刊行物「Communications and Networking for the IBM PC」に記載されている。本発明は、上述した刊行物に記載されている技術を対象としているけれども、イーサネットＬＡＮ及びＩＢＭのＳＮＡ以外の他のネットワークによって相互接続されたＩＢＭのＲＩＳＣベースのパーソナルコンピュータ以外の他の異なったコンピュータを用いて実施することができることには注意を払う必要がある。従って、本発明は、例えば、ＯＳ／２オペレーティング・システムの下で動作するＩＢＭ社のＰＳ／２において実施することができる。ＰＳ／２コンピュータ及びＯＳ／２オペレーティング・システムに関する情報については、ＩＢＭ社で刊行した刊行物「Technical Reference Manual Personal Systems/2 Model 50, 60 Systems, Part No. 68x2224 — Order Number S68X-2224」と、「OS/2 2.0 Technical Library, Programming Guide Volume 1~3 Version 2.00 — Order No. 10G6495 及び10G6494」とを参照されたい。

【００１９】図１に戻って説明を続けると、クライアント１０及びサーバ１２の各々は、そのコンピュータがサービスを要求しているのか、またはそのコンピュータがサービスを提供するのかに応じて、クライアントとしてか、またはサーバとして動作する。通常、クライアントは、遠隔プロシージャ呼び出し（ＲＰＣ）を用いてサーバ・プロセスにサービスを要求する少なくとも１つのプロセスを含んでいる。然しながら、多くのＤＣＥアプリケーションは、「クライアント」プロセス及び「サーバ」プロセスの両方が同じホスト・コンピュータ中で実行されるような態様で書かれている。この態様は、クライアント／サーバ１５によって図１に示されている。クライアント／サーバ１５中のサーバ・プロセスのために意図されたクライアント・プロセスからのＲＰＣは、下記の説明を行なう目的のために、「ローカル」または「ローカルＲＰＣ」と呼ぶ。

【００２０】従来のＯＳＦ－ＤＣＥ－ＲＰＣ機構において、ローカルであると否とに拘らずすべてのＲＰＣは、「ncacn_ip_tcp」か、または「ncadg_ip_udp」プロトコル順序であるネットワーク・ベース（例えば、「ｉｐベース」）プロトコル順序を用いて遂行される。この場合、「ncacn」は接続指向のＯＳＦ－ＤＣＥ－ＲＰＣプ

ロトコル順序を表わし、そして「ncadg」は非接続のＯＳＦ－ＤＣＥ－ＲＰＣプロトコル順序を表わす。これらのプロトコルを用いた時、「ｉｐ」術語によって明らかなように、インターネット・アドレス・ファミリィ（ＡＦ＿ＩＮＥＴ）通信ドメインが使用される。各プロトコル順序の最後の構成要素はソケット・タイプを参照するから、従って、例えば、「udp」はデータグラムを参照する。ＲＰＣに応答して、セル・ディレクトリ・サービス・デーモン（cell directory service daemon－ＣＤＳＤ）の呼び出しは、サーバ・プロセスが実行されているネットワーク・アドレス及びポート番号としてのサーバ・プロセスの位置を特定する、所謂、「バインディング・ハンドル」を戻す。また、バインディング・ハンドルは、ＲＰＣによって要求されたサーバ・プロセスを識別する。次に、バインディング・ハンドルは、クライアント・プロセス及びサーバ・プロセス間の通信路を定義するために、ＲＰＣ機構によって用いられる。この通信路は、ソケットをオープンするために、ＡＦ＿ＩＮＥＴのｉｐベース（即ちネットワーク層）のプロトコル順序を用いて定義される。この通信路は、クライアント・プロセスからトランスポート層及びネットワーク層を通ってネットワークに出た後に、サーバ・プロセスが動作しているホスト・コンピュータに関連された層に戻るループを形成する。

【００２１】従って、クライアント・プロセス及びサーバ・プロセスが同じホスト・コンピュータ中に存在しているとしても、トランスポート層（ＴＣＰまたはＵＤＰ）及びネットワーク（ＩＰ）層がループバック・メッセージを送るのに使用される。これは、図２に示されている。

【００２２】本発明に従うと、ｉｐベースのプロトコル順序を使用する代りに、クライアント・プロセスに対して透過的な態様（影響を与えない）でローカルＲＰＣを助長するための代替プロトコル順序が用いられる。この透過的な態様のために、ローカルＲＰＣは、アプリケーションに「隠されて」遂行されると言われる。若しオペレーティング・システムがＡＩＸ、またはＯＳ／２オペレーティング・システムであれば、プロトコル順序「ncacn_unix_stream」は、接続指向ｉｐベースのプロトコル順序（即ち、「ncacn_ip_tcp」）を識別するローカルＲＰＣに使用される。指定子「ncacn」は、接続指向ＲＰＣプロトコルを指定し、「unix」はＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）通信ドメインを識別し、そして「stream」はＵＮＩＸドメインのストリーム・ソケットを識別する。ＯＳ／２オペレーティング・システムの下でローカルＲＰＣ機能が動作している場合、非接続ｉｐベースのプロトコル順序（即ち、「ncadg_ip_udp」）の下で発行されたＲＰＣは、プロトコル順序「ncadg_unix_dgram」の使用を生じる。この場合、「ncadg」は非接続ＲＰＣプロトコルを表わし、そして「unix_dgram」はＵＮＩＸドメイン・データグラム・ソケットを識別する。勿論、上述したオペレーティング・システムは、単なる例示であって、本発明はこれらのオペレーティング・システムのみに限定して解釈されるべきものではない。何れの場合でも、クライアント・プロセスには、実際のＲＰＣが、異なったローカルの機構を用いて遂行されたとしても、クライアント・プロセスが予期するプロトコル順序が戻される。使用される実際のプロトコル順序は、「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）」を使用することによって、トランスポート層及びネットワーク層の使用を回避する。

【００２３】ＡＦ＿ＵＮＩＸを使用することによって、オペレーティング・システム・カーネルは、同じホスト・コンピュータ中の２つのプロセスの間の通信を連結するタスクを操作する。ＡＦ＿ＵＮＩＸは、クライアント・プロセスとサーバ・プロセスとの間のインタープロセス通信（ＩＰＣ）を実行するために好ましい技術であるけれども、他のオペレーティング・システム（ＯＳ）のＩＰＣ機構（共有メモリとかメッセージ待ち行列など）も同じように使用することができる。どのような場合でも、ローカルＩＰＣ機構の使用は、トランスポート層及びネットワーク層を通したループバック呼び出しを行なうよりも著しく速いので、性能の改善は顕著なものがある。本発明の機能は図３に示されている。

【００２４】ncacn_unix_streamプロトコル順序は、同じホスト・コンピュータ中に存在するＤＣＥ（分散計算環境）アプリケーションのプロセスの間で行なわれるＲＰＣによってのみ用いられるのが好ましい。ＲＰＣランタイム（runtime）は、若しサーバが同じホスト・コンピュータ中に存在しなければ、クライアント・アプリケーションに、ncacn_unix_streamのバインディングを戻さない。ncacn_unix_streamプロトコル順序は、同じホスト・コンピュータ中のＤＣＥアプリケーションによって使用されるものだから、これらのバインディングはネットワーク・アドレスを含んでいない。また、ncacn_unix_streamのバインディング・ハンドルのエンドポイント（endpoint）は、ＵＮＩＸソケット・ファイルのフル通信路名（full pathname）として表示される。固有のソケット・ファイルが、クライアント・プロセスとサーバ・プロセスとの間で設定された各アソシエーション（組み合せ）のために使用される。デフォルトによって、これらのソケット・ファイルは、接頭部：DCE_CNを持つファイル名を有するディレクトリ、/var/dce/rpc/socket中にオープンされる。また、デフォルトによって、各ソケット・ファイルの名前は、オブジェクトＵＵＩＤ（universal unique identifier－ユニバーサル・ユニーク識別子）であり、このユニバーサル・ユニーク識別子は各ファイル名の固有性を保証する。ＵＵＩＤは、ＵＵＩＤゼネレータ・ルーチンによって作成された

実質的に長くランダムな数字である。ＲＰＣランタイムによって、エンドポイントをncacn_unix_streamのバインディングに割り当てる時、そのバインディングは、「オブジェクトＵＵＩＤ」であり、これは、ソケット・ファイル名の固有性を保証する。これは、ＤＣＥアプリケーションの２つの呼び出しにおいて、ソケット・ファイルが再度使用されるような場合が皆無であることを意味する。

【００２５】ＵＮＩＸベースのオペレーティング・システムにおいて、ＯＳカーネルは、そのカーネルがユーザのために作業を行なう前に、ファイルをオープンすることをユーザに要求する。本発明によって作成されたソケット・ファイルは、ＯＳカーネルのプレースホルダーとして保存される長さのないファイルである。固有の各ソケット・ファイルは、作成されたファイル・システム中にｉノードのエントリを占領する。ＯＳカーネルは、ファイル・システム中のプレースホルダーとしてソケット・ファイルを用いることによってＲＰＣを遂行し、次に、それ自身のデータ構造を制御して、ＲＰＣを具現させるために、クライアント・プロセス及びサーバ・プロセスの間でデータを送信し、かつ受信する。従って、ＯＳカーネルは、この目的のために、例えば「Ｍｂｕｆ」のデータ構造を使用することができる。

【００２６】下記の記載は、本発明において使用されるncacn_unix_streamストリングのバインディングの幾つかの例を示している。

ncacn_unix_stream:[]

ncacn_unix_stream:[/var/dce/rpc/socket/0063980e357b-1e07-878b-10005a4f3bce]

このように、プロトコル順序は、接続指向の「ncacn」ＲＰＣプロトコル、AF_UNIX通信ドメイン（「ｕｎｉｘ」のコンポーネントによって識別される）及び「stream」タイプのソケットの返却を含んでいる。１番目の例示において、エンドポイント（つまり、括弧［］内のデータ）は空である。２番目の例示においては、エンドポイントは、デフォルトのソケット・ディレクトリ「/var/dce/rpc/socket/_」及びＵＵＩＤを識別する。

【００２７】従って、エンドポイントは、/var/dce/rpc/socketディレクトリ中のオブジェクトＵＵＩＤである必要はない。ＩＤＬ（Interface Definition Langauge-インターフェース定義言語）仕様書のファイル中の「エンドポイント」の属性は、固有のファイルに対する通信路名を特定するのに用いることができる。あるいは、アプリケーションは、ncacn_unix_streamストリングのバインディングを、コマンド・ライン引数として取ることができ、そして、その引数を、rpc_binding_from_string_binding()ルーチンを用いてバインディング・ハンドルに変換することができる。ユーザにより特定されたソケット・ファイル通信路の例を下記に示す。

ncacn_unix_stream:[/var/dce/rpc/socket/foo]

ncacn_unix_stream:[/tmp/6094]

ncacn_unix_stream:[/5423]

最後に示した例において、エンドポイントは絶対通信路によって表示されていないので、ncacn_unix_stream:[/var/dce/rpc/socket/5423]が仮定され、ソケット・ファイルを作成するために使用される。また、エンドポイント・マップのエントリは、絶対通信路を持っている。このことは、相対通信路が初めて使用される前に、相対通信路は、絶対通信路まで常に拡張され得るので、２つのフォームが相互に交換可能に用いられた時に問題を生じることがない。

【００２８】ローカルＲＰＣ技術のオリジナルの実施において、ＲＰＣランタイム（実行時）ライブラリは、クライアント及びサーバが同じホスト・コンピュータ中に存在する場合を認識し、そして、機能rpc_ns_binding_import_next()が他のプロトコル順序のためのバインディングを戻す前に、そのようなバインディングを、機能rpc_ns_binding_import_next()に戻させることによるか、あるいは、バインディング・ベクトルの低い位置中のすべてのncacn_unix_streamのバインディングを、機能rpc_ns_binding_lookup_next()に戻させることによってncacn_unix_streamの使用を促進する。性能上の理由のために、そのような実施例においてrpc_ns_binding_<import/lookup>_next()ルーチンから戻されたncacn_unix_streamのバインディング・ハンドルは、完全に結合される。これは、そのバインディングのベクトルがＲＰＣエンドポイント・マップから到来し、そのバインディングの中にエンドポイントを持っていたことによる。この実施例において、ncacn_unix_streamのバインディング情報は、ＲＰＣのＡＰＩルーチンrpc_ns_binding_export()が呼び出された時、ＣＤＳ（セル・ディレクトリ・サービス）ネーム・スペースのデータベースへエクスポートされない。この理由は、クライアント及びサーバのアプリケーションが同じホスト・コンピュータ中に存在する時だけに、ncacn_unix_streamのバインディングが使用可能だからである。ＣＤＳネーム・スペースは、「ＤＣＥセル」中の任意のコンピュータ中のＤＣＥのアプリケーションによって読み取ることができるから、ＣＤＳネーム・スペースのデータベース中のncacn_unix_streamストリームのバインディングは、遠隔地のサーバと通信する必要のあるクライアントに対して使用不能である。

【００２９】その代わりに、ncacn_unix_streamのバインディングは、ＤＣＥのサーバ・アプリケーションがＲＰＣのＡＰＩルーチンrpc_ep_register()を呼び出した時に、ＲＰＣエンドポイント・マップのデータベース中にのみ登録される。この理由は、ＤＣＥセル中の各ホスト・コンピュータで実行するＲＰＣエンドポイント・マップのデーモンがあるためであり、これは、ホスト・コンピュータ特定の情報をストアするために、更に良い論理的位置にする。クライアントが、互換性を有するサー

バのバインディング用のＣＤＳネーム・スペースを問い合わせた時、ＲＰＣランタイムは、セルがローカル・ホスト・コンピュータ中のサーバ用であるか否かを決定する。若しセルがサーバ用であれば、ＣＤＳネーム・スペースから、すべてのバインディングを戻すことに加えて、ＲＰＣランタイムは、互換性を有するncacn_unix_streamのバインディングがあるか否かを検出するためにＲＰＣエンドポイント・マップに行く。若し互換性を持つncacn_unix_streamのバインディングがあれば、そのバインディングはＣＤＳネーム・スペースから戻されたバインディングよりも高い優先権を与えられる。若しクライアント及びサーバが異なったホスト・コンピュータの中にあれば、エンドポイント・マップの検索は行なわれない。何故ならば、この場合のエンドポイント・マップの検索は、使用可能なバインディングを生じないからである。

【００３０】ＤＣＥサーバ・アプリケーションが通常の状態で存在する時、この状態は、ＤＣＥサーバ・アプリケーションが存在する前に、ＲＰＣのエンドポイント・マップから、他のものと共に、そのエンドポイントを登録しない。ncacn_ip_tcp及びncacn_ip_udpのようなｉｐベースのプロトコルの場合において、ＤＥＣアプリケーションを受信しようとしていたポートは、オペレーティング・システムによって整理されるから、ポートを解放するためのアプリケーション動作は要求されない。ncacn_unix_streamのエンドポイントは、ユーザ・スペース・ファイルであり、これらのファイルは、ＤＣＥアプリケーションが存在する時には、オペレーティング・システムによって除去されない。

【００３１】既に説明したように、ＲＰＣランタイムによって、エンドポイントをncacn_unix_streamのバインディングに割り当てる時に、このバインディングは、固有であることが保証されている「オブジェクトＵＵＩＤ」である。このことは、ＤＣＥアプリケーションの２つの呼び出し中で、ソケット・ファイルが再度使用されることはあり得ないことを意味する。従って、古いソケット・ファイルが残留している事実は、短時間では問題にはならない。然しながら、時間の経過と共に、これらのソケット・ファイルは累積されるであろう。既に述べたように、ソケット・ファイルは、長さがゼロであるけれども、しかし、各ソケット・ファイルは、ソケット・ファイルが作成されたファイル・システム中にｉノード・エントリを占める。従って、これらの古いソケット・ファイルを整理するための何らかの手段を持つことが必要である。これは、ＲＰＣエンドポイント・マッパ（ＲＰＣデーモン）によって行なわれる。

【００３２】/var/dce/rpc/socketディレクトリ中のソケットだけが自動的に整理されることには注意を払う必要がある。若しＤＣＥアプリケーションがＤＣＥアプリケーション以外のディレクトリを用いて、そのncacn_unix_streamエンドポイントを作成したならば、古いエンドポイントが整理されることを保証することは、アプリケーション、またはアプリケーションのユーザの責任である。これは、rpc.cleanユティリティを用いて行なわれる。rpc.cleanユティリティは、検査されるべき古いソケット・ファイルに関するディレクトリを、ユーザにより特定させる。このユティリティは、そのディレクトリを検索し、そして検出されたすべての古いソケット・ファイルを除去する。これは、ＲＰＣのエンドポイント・マップが行なう整理と同じ整理を遂行する。ＤＣＥ管理記述、dce.clean、rc.dce、またはrmdceが実行される時には常に、ソケット・ファイルの累積問題を解決する付加的な保護策として、rpc.cleanユティリティが含まれる。

【００３３】ＤＣＥの多くの内部活動（即ち、ＲＰＣデーモン及びセキュリティ／ネーム・スペースのサービスを含む通信活動）が、同じコンピュータにおいて実行されている２つのプロセスの間でＲＰＣを用いて遂行された場合であって、上述のローカルＲＰＣのアプローチ（ｕｎｉｘドメイン・ソケットを用いた方法）が実施された場合、ＤＣＥシステム全体の性能は約４０％向上した。このローカルＲＰＣ構造は、本発明に従って更に改善される。本発明の実施例において理解できるように、すべてのローカルＲＰＣ動作はデフォルト動作として最適化されることが望ましい。明示的な接続指向（ＣＮ）プロトコルは、ＲＰＣのアプリケーション及びクライアント・プロセスに対して透過的な態様（影響を与えない）で、しかも、速いローカルunix_streamプロトコルに自動的に切換えられる。同様に、明示的な非接続（ＤＧ）プロトコルは、速いローカルunix_dgramプロトコルに自動的に切換えられる。本発明の実施例において、明示的なunix_streamプロトコルをクライアント・プロセスに戻すことは最早や必要としない。プロトコルの切り換えは、ＲＰＣ機構及びアプリケーションに「隠されて」遂行され、これは性能を更に向上させる。

【００３４】特に、上述したオリジナルのローカルＲＰＣアプローチにおいて、unix_streamプロトコルは、明示的に与えられており、ＤＣＥデーモンの「エンドポイント・マッパ」中に現われる異なったタイプのエンドポイントの形式でユーザに可視的である。アプリケーションが、接続指向プロトコル順序かまたは非接続プロトコル順序を使用するように選択し、かつローカルＲＰＣ機構が暗黙的に優先権を取る場合か、または、アプリケーションが、unix_streamプロトコルを使用するように選択した場合に、改良型のローカルＲＰＣが行なわれる。改良型のローカルＲＰＣは、unix_streamのエンドポイントのための動作に関連された「エンドポイント・マッパ」を除去する。

【００３５】従って、本発明の上述の改良型の実施例において、透過的な態様の切換えはユーザのプロトコル順

序（ＲＰＣにおいて識別される）から、ある種のローカル・プロトコルに移り、そして、この透過的な態様の切換えは自動的に生じるのが望ましい。このアプローチは、明示的なunix_streamまたはunix_dgramプロトコルを取り除き、そして、接続指向（ＣＮ）及び非接続（ＤＧ）プロトコルのために最適化されたローカルＲＰＣをデフォルトとして設定する。然しながら、ＣＮ及びＤＧデフォルトの意味を保存するために、別個のＣＮローカルＲＰＣ内部サポート（ncacn_unix_stream）及びＤＧローカルＲＰＣ内部サポート（ncacn_unix_dgram）が作成される。

【００３６】クライアント・プロセスが遠隔プロシージャ呼び出しを行なった時に、分散計算環境内のクライアント・プロセス及びサーバ・プロセス間の通信を管理する方法における主要なステップを示した図４の流れ図において、上述の代替プロトコル順序の処理方法が示されている。この方法において、クライアント・プロセスは、ホスト・コンピュータがＡＩＸのようなＵＮＩＸベースのオペレーティング・システムをサポートしており、かつ、トランスポート層及びネットワーク層を有する物理的ネットワークに接続されている同じホスト・コンピュータ中に常駐しているものと仮定する。この方法は、サーバ・プロセスが同じホスト・コンピュータ中に存在しているか否かを決定するステップ３０で開始する。若しサーバ・プロセスが同じホスト・コンピュータ中にあれば、この方法は、遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を持つバインディング・ハンドルをクライアント・プロセスへ戻すステップ３２に続く。次に、クライアント・プロセスは、予期しているプロトコル順序（ＲＰＣで与えられたプロトコル順序）を「検出」する。次に、この方法は、遠隔プロシージャ呼び出しと通常関連しているプロトコル順序を、クライアント・プロセス及びサーバ・プロセス間のインタープロセス通信路を設定する代替プロトコル順序にマップ処理するステップ３４に続く。（ステップ３２及び３４を反対にすることができる。）従って、例示的な説明目的に限定して、若し遠隔プロシージャ呼び出しに関連したプロトコル順序が接続指向のプロトコル「ncacn_ip_tcp」であれば、このステップは、そのプロトコル順序を、同じコンピュータ上で実行している代替プロトコル順序にマップする。この機能は常に使用可能にされていることが望ましい。従ってサーバ・プロセス及びクライアント・プロセスが同じコンピュータ中に存在する場合には、「ncacn_ip_tcp」プロトコル順序の代わりに「ncacn_unix_stream」プロトコル順序が使用され、そして、「ncadg_ip_udp」プロトコル順序の代わりに「ncadg_unix_dgram」プロトコル順序が使用される。上述した２つの代表的なオペレーティング・システムのコンテキストにおいて、この技術は、ＡＩＸ及びＯＳ／２の両方のオペレーティング・システムに対して、「ncacn_ip_tcp」プロトコル順序を「ncacn_unix_stream」に切換え、そして、ＯＳ／２オペレーティング・システムだけに対して、この機能が動作可能にされる時に「ncacn_ip_udp」プロトコル順序を「ncadg_unix_dgram」に切換えることが望ましい。ＯＳ／２オペレーティング・システムだけに限定した後者の場合の理由は、「ncadg_unix_dgram」プロトコル順序を実行することが、ＡＩＸオペレーティング・システムの性能を顕著に改善しないからである。従って、若しアプリケーションがＡＩＸの「ncadg_ip_udp」プロトコル順序を使用したならば、ローカルＲＰＣは、サーバ及びクライアントのプロセスが同じコンピュータ中に存在したとしても推奨されるものではない。この機能を使用不能にするために、環境変数、DISABLE_AUTO_LRPCは１にセットされる。

【００３７】以下の記載は、最適化されたローカルＲＰＣ機構の細部を更に詳しく説明したものである。この説明を総括的に言えば、ＤＣＥのプログラミング規則に精通していること、より特定して言えば、ＲＰＣ機構に精通していることが前提とされる。設計の第１の局面は、どのプロトコル順序がサポートされているかをユーザが識別することであり、この識別はＲＰＣ初期化ルーチンの間で行なわれる。若しユーザがＲＰＣ＿ＳＵＰＰＯＲＴＥＤ＿ＰＲＯＴＳＥＱＳを、１つまたはそれ以上のプロトコル順序にセットしたならば、これらのプロトコル順序だけがサポートされる。然しながら、ＲＰＣ初期化ルーチンを終了する時に、ＤＩＳＡＢＬＥＤ＿ＡＵＴＯ＿ＬＲＰＣ変数がチェックされる。若しこの変数が１にセットされていなければ、大域変数「defaultrpc」が「真」にセットされる。次に、ＲＰＣ「ランタイム」は最適化されたローカルＲＰＣ機能をサポートする。これは、「ncacn_unix_stream」及び「ncadg_unix_dgram」プロトコル順序がＲＰＣ「ランタイム」によってサポートされていることを意味する。「ランタイム」コードのすべての変更は、defaultlrpc変数が「真」である時だけに実行される。

【００３８】最適化されたローカルＲＰＣ機能をサポートするために必要な変更であって、アプリケーションのサーバ・プロセスに関連する変更は、以下に説明する通りである。以下の説明は、ncacn_ip_tcpプロトコル順序の場合に関連するものであるけれども、同じ考え方をncacn_ip_udpプロトコル順序に適用することができるのは理解されるべきである。ローカルＲＰＣの１つの機能は、サーバ・プロセスが聴取しているプロトコル順序用の公知のエンドポイント及び動的エンドポイントのすべてのエンドポイントを含む「Ｕｎｉｘドメイン」ソケットをオープンしなければならないことである。この理由は、若しストリング・バインディングが公知のエンドポイントまたは動的エンドポイントを用いてクライアントに与えられたならば、ＤＣＥデーモンが同じコンピュータ中で実行されるものと推測することができないからで

ある。従って、ncacn_unix_streamプロトコル順序及び特別のエンドポイントを用いて、完全な宛先rpcアドレスを作成し、次に、その宛先rpcアドレスをバインディング表示中にストアする必要がある。これは、「Ｕｎｉｘドメイン」ソケットに結合されている特別な公知のエンドポイント及び動的エンドポイントのすべてを持っている「Ｕｎｉｘドメイン」ソケットを、サーバ・プロセスが聴取しなければならないことを意味する。

【００３９】最適化されたローカルＲＰＣを実施するために、ファイルsrc/rpc/runtime/cnnet.cの中のrpc_cn_network_use_protseq()は変更されねばならない。このルーチンにおいて、ｔｃｐソケットがオープンされる時には常に、対応するｕｎｉｘストリーム・ソケットがオープンされる。この処理が、図４に示したマップ作成ステップである。このｕｎｉｘストリーム・ソケットのために、ｕｎｉｘのrpc_addrが割り当てられ、tcpエンドポイントが命名規則に従ってｕｎｉｘストリームのエンドポイントに変換される。また、このｕｎｉｘストリーム・ソケットは、ソケット記述子に加えられる。従って、rpc_cn_network_use_protseq()ルーチンは以下のように実行される。

if auto LRPC is on（若し自動ＬＲＰＣがオンならば）
Get the tcp endpoint（ｔｃｐエンドポイントを獲得せよ）
Allocate memory for the new unix RPC address（新しいｕｎｉｘＲＰＣアドレス用のメモリを割り当てよ）
Insert the protseq id, socket address structure length,and Network- Address Family into RPC addressstructure（protseq識別子、ソケット・アドレス構造の長さ及びネットワーク・アドレス・ファミリィをＲＰＣアドレス構造中に挿入せよ）
Open a unix socket and bind the RPC address to it（ｕｎｉｘソケットをオープンし、それにＲＰＣアドレスを与えよ）
Add the socket to the socket descriptor.（ソケットにソケット記述子を加えよ。）
end if

以上の説明は、アプリケーション・サーバ・プロセスに関係した変更に関するものである。

【００４０】アプリケーションのクライアント・プロセスに関係した変更を以下に説明する。クライアントのアプリケーションがＲＰＣランタイムに対して先ず最初に行なう第１の呼び出しは、rpc_call_start()であり、この呼び出しは、転じて、バインディング・ハンドル中のプロトコル識別子に従って、プロトコル特定のルーチン、つまりrpc_dg_call_start()か、またはrpc_cn_call_start()を呼び出す。プロトコル特定のルーチンを呼び出す前に、rpc_call_start()に対して下記のような変更が行なわれる。

【００４１】このルーチンは、バインディング・ハンドル中の宛先アドレスがローカル・ホスト・コンピュータのipアドレスの１つと等しいか否かを決定することによって開始する。（この比較は、若しＴＣＰ／ＩＰがコンピュータ中に導入されていなければ動作しない。むしろ、ＮｅｔＢｉｏｓ用のＲＰＣプロトコル順序に関係したアドレスのために、異なったタイプの比較が必要とされる。）若しこの比較が真であれば、アプリケーションがrpc_call_start()に通されたプロトコル順序に代わって、ncacn_unix_streamプロトコル順序が使用されるように、バインディング表示中の該当するフィールドが変更される。バインディング表示は下記のフィールドを持っている。

```
typedef struct {
rpc_list_t link;
rpc_protocol_id_t protocol_id;
signed8 refcnt;
uuid_t obj;
rpc_addr_p_t rpc_addr;
unsigned is_server: 1;
unsigned addr_is_dynamic: 1;
rpc_auth_info_p_t auth_info;
unsigned32fork_count;
unsigned bound_server_instance: 1;
unsigned addr_has_endpoint: 1;
unsigned32 timeout;
signed8 calls_in_progress;
pointer_t ns_specific;
rpc_clock_t call_timeout_time; }
rpc_binding_rep_t, rpc_binding_rep_p_t;
```

このルーチンに従って、バインディング表示中のプロトコル識別子フィールドは、接続指向ＲＰＣプロトコル識別子、つまり０で置き換えられる。バインディング表示は、その中にストアされた宛先ｒｐｃアドレスを持っており、そしてrpc_addrは、rpc_addr_p_tタイプである。

```
typedef struct {
rpc_protseq_id_t rpc_protseq_id;
unsigned32 len;
sockaddr_t sa;
} rpc_addr_p_t;
```

【００４２】ファイルsrc/rpc/runtime/cncall.c中の機能rpc_cn_call_start()は変更されなければならない。先ず、元のrpc_addr構造が保存される。この時点において、rpc_addrは公知のエンドポイントを持っている。ルーチンは、このエンドポイントをｕｎｉｘストリーム・エンドポイントに変換した後にｕｎｉｘストリームのための宛先rpcアドレスを作成する。次に、ルーチンは、バインディング表示中のrpc_addrを、新しいrpc_addrに置き換える。ncacn_unix_streamプロトコルを持つこのrpc_addrは「エンドポイント・マッパ」に登録されず、従って、ユーザに対して可視的ではない。その後、ＲＰ

Cはｕｎｉｘストリーム及び接続指向ＲＰＣプロトコルを使用して可視的になる。すべての送信及び受信が完了した後、元のrpc_addr構造は、バインディング表示にコピーされ、そしてクライアント・プロセスに戻される。rpc_cn_call_start()の機能は下記の通りである。

if auto LRPC is on（若し自動ＬＲＰＣがオンならば）

Allocate memory for the new unix RPC address（新しいｕｎｉｘＲＰＣアドレス用のメモリを割り当てよ）

Insert the protseq id, socket address structure length,and Network- Address Family into RPC addresss tructure（protseq識別子、ソケット・アドレス構造の長さ及び「ネットワーク・アドレス・ファミリィ」をＲＰＣアドレス構造中に挿入せよ）

Save the original rpc_addr from the bindingrepresentation（バインディング表示からオリジナルのＲＰＣアドレスを保存せよ）

Copy the unix rpc_addr to the binding representation（ｕｎｉｘＲＰＣアドレスをバインディング表示にコピーせよ）

Do the transmits and receives（送信及び受信を行なえ）

Copy the original rpc_addr back to the bindingrepresentation（オリジナルのＲＰＣアドレスをバインディング表示にコピーし戻せ）

If any error occured in the transmit or receive usingunix stream,try again transmitting/receiving using the originalrpc_addr（若しｕｎｉｘストリームを用いた送信または受信においてエラーが発生したならば、オリジナルのＲＰＣアドレスを用いて送信／受信を試行せよ）

Return to the client（クライアントに戻れ）

end if

【００４３】最適化されたローカルＲＰＣ機構において、ｕｎｉｘストリームを持つバインディングは、「エンドポイント・マッパ」中に登録されず、あるいはネーム・スペースにエクスポートされない。クライアント・プロセスが、import_begin及びimport_next呼び出し機能を用いてバインディングに問い合せを行なった時、クライアント・プロセスは、ncacn_unix_streamを持つバインディングを獲得しない。ＡＰＩ呼び出しrpc_server_inq_bindingsは、ＲＰＣが行なわれるサーバのバインディングを戻す。この機能はバインディングを得るために、聴取側のソケットからのソケットを使用する。サーバはストリーム・ソケットにおいて聴取しているから、下記のフィルタが与えられる。

for loop for enquiring the bindings from the listener sockets（聴取ソケットからのバインディングの問い合せを行なうためのループ）

if the socket protseq is unix stream,（若しソケットprotseqがｕｎｉｘストリームならば、）

skip the binding（そのバインディングをスキップせよ）

end for loop（ループの終り）

【００４４】既に説明したように、本発明に従って、アプリケーションのクライアントは、ｕｎｉｘストリーム・プロトコルを使用してアプリケーションのサーバと対話するから、従って、このプロトコルはアプリケーションのサーバに露呈される。これを遮蔽するために、rpc_binding_from_string_bindingルーチンが変更される。この場合、rpc_addrは、バインディング・ハンドルからのものではなく、ローカル・コンピュータのアドレス及びncacn_ip_tcpプロトコル順序によって満たされる。これを行なうことによって、クライアント・プロセスは、それが予期していたプロトコル順序を「検知」し、そして代替プロトコル順序が通信路に影響するように使用される。その結果、ncacn_unix_streamが、ユーザから隠蔽される。この機能は下記の通りである。

if the binding rep rpc_addr has unix stream in it（若しバインディング表示rpc_addrがその中にｕｎｉｘストリームを持っているならば）

enquire the local machine address with tcp protocolsequence（ローカル・コンピュータ・アドレスをｔｃｐプロトコル順序に問い合せよ）

Insert the tcp protseq id, socket address structurelength, and IP- Network Address family into RPCaddress structure（tcp Protseq識別子、ソケット・アドレス構造の長さ及びＩＰネットワーク・アドレス・ファミリィをＲＰＣアドレス構造に挿入せよ）

Copy this rpc_addr to the binding representation（このrpc_addrをバインディング表示にコピーせよ）

return this rpc_addr back（このrpc_addrを戻せ）

end if

【００４５】上述した設計は、最適化されたローカルＲＰＣ構造を実行するために必要な変更を説明するものである。この改善の重要な目標は、この最適化されたＲＰＣ構造の実施がユーザに不可視的にすることと、従来のクライアント／サーバ・アプリケーションに対して逆向きの互換性を持たせることとを保証することである。

【００４６】ncacn_unix_streamプロトコル順序に対してどのようにしてＲＰＣのサポートを付加するかについての細部の説明を以下に記載する。

【００４７】0.1.1　外部インターフェースの説明

ＲＰＣランタイム及びＲＰＣＤの外部インターフェースは以下のように変更される。ユーザは、従来からサポートされているプロトコル順序に加えて、付加的なプロトコル順序を指定するための新しい能力を持っている。この新しいプロトコル順序は「ncacn_unix_stream」と名付けられる。ＡＰＩ機能パラメータ、環境変数、またはコマンド・ライン引数を、プロトコル順序用ストリング・フォーマットの仕様に用いることができる場合には常

に、ストリーム「ncacn_unix_stream」の使用が可能である。

【００４８】0.1.2　内部設計の説明

ＲＰＣの拡張は、クライアント及びサーバが同じホスト・コンピュータ中に存在する場合に使用される新しいプロトコル順序（ncacn_unix_stream）のためのサポートを加える。

【００４９】このＲＰＣの拡張は、ユーザに対して仮想的に不可視であり、そして、従来のクライアント／サーバ・アプリケーションとは逆向きの互換性を有する。

【００５０】本発明は、ＩＢＭのＯＳ／２プラットホームのような他のプラットホームに対して移植性（portable）がある。ＯＳ／２はＵＮＩＸドメインに対するサポートを有する（ＭＰＴＳを介して）。ＯＳ／２の「共通ポーティング・プラットホーム（ＣＰＰ）」のＤＬＬは、ＯＳ／２を変更することなく、殆どすべてのＵＮＩＸ特定サブルーチン呼び出しを使用可能にする。

【００５１】新しいプロトコル順序の付加が従来のコードを殆ど変更することなく、ＲＰＣランタイム・ライブラリの設計を行なうことができる。これは、２つの手段、即ち（１）モジュラー・データ構造と、（２）エントリ・ポイント・ベクトルとを使用することによって達成される。モジュラー・データ構造は、プロトコル順序を作る幾つかの要素に関する一貫した機構を与えることによって新しいプロトコル順序を付加することを容易にする。ＲＰＣランタイム・コードの大部分がすべてのプロトコルに共通なので、エントリ・ポイント・ベクトルを使用することは、インデックスを通して機能アレイのライン毎に、プロトコル特定のルーチンにアクセスすることができる。プロトコル識別子は、エントリ・ポイント・ベクトル中のインデックスとして使用される。新しいプロトコル順序には、そのプロトコル順序に特定された処理を操作するための新しいルーチンを加えることが必要である。

【００５２】ncacn_unix_streamプロトコル順序は、AF_＊UNIXアドレス・ファミリィと、ＳＯＣＫ＿＿ＳＴＲＥＡＭのソケット・タイプとを用いて作成される。ソケット・アドレスは、インタープロセス通信機構としてのフル通信路名の仕様を用いて、構造sockaddr_unを使用する。クライアント及びサーバはこのプロトコル順序を用いるために同じホスト・コンピュータ中に常駐していなければならない。このプロトコル順序は接続指向であり、そしてＣＮプロトコル識別子を使用する。

【００５３】以下の項において、新しいプロトコル順序を実行するために変更を必要とするＲＰＣソース・ファイルを説明する。以下の説明において、広範な含意変更（implication change）から開始して、より特別な変更に減縮するようなファイルのリストを作るように試みられている。

【００５４】0.1.2.1　src/rpc/runtime/com.h

このファイルは既存のファイルである。下記のような定数識別子を付加することが必要である。これらの定数は、使用する構造及び適正なインデックス・エントリ・ポイント・ベクトルにアクセスするために、すべてのＲＰＣランタイム・コードに亙って使用される。

ＲＰＣプロトコル順序識別子定数：

```
#define rpc_c_protseq_id_ncacn_unix_stream 5
```

ＲＰＣプロトコル順序ストリング定数：

```
#difine rpc_protseq_ccacn_unix_stream "ncacn_unix_
stream"
```

ＲＰＣ「ネットワーク・アドレス・ファミリィ」定数：

```
#define rpc_c_naf_id_unix 1
```

【００５５】0.1.2.2　src/rpc/runtime/comp.c

このファイルは既存のファイルである。下記に示すテーブル・エレメントを付加することが必要である。これらのテーブル・エレメントは、「Ｕｎｉｘドメイン・アドレス・ファミリィ」と、ncacn_unix_streamプロトコル順序とのためのサポートを付加する。

【００５６】大域アレイrpc_q_protseq_id[]は下記のように拡張された。

```
GLOBAL rpc_protseq_id_elt_t rpc_q_protseq_id[rpc_protseq_id_max] =
{
    .../* existing table elements */ （存在するテーブル・エレメント）
    {
            0,
            rpc_c_protseq_id_ncacn_unix_stream,
            rpc_c_protocol_id_ncacn,
            rpc_c_naf_id_unix,
            rpc_c_network_protocol_id_uns /* Unspecified */
            rpc_protseq_ncacn_unix_stream
    }
{;
```

【００５７】大域アレイrpc_q_naf_id[]は下記のように※　※拡張されなければならない。

```
GLOBAL rpc_naf_id_elt_t rpc_q_naf_id[rpc_naf_id_max] =
{
```

```
                25                                                    26
                    ... /* existing table elements */ （存在するテーブル・エレメント
            )
                    {
                            rpc_unix_init,
                            rpc_c_naf_id_unix,
                            rpc_c_network_if_id_stream,
                            NULL
                     }
                     .../* existing table elements */ （存在するテーブル・エレメント
            )
              };
```

【００５８】0.1.2.3 src/rpc/runtime/unixnaf.c
このファイルは作成を必要とする新しいファイルである。このファイルは、「Ｕｎｉｘドメイン・ネットワーク・アドレス・ファミリィ」のために必要なエントリ・＊ポイント・ベクトルのルーチンのためのサポートを与える。下記のように定義されたrpc_naf_epv_tと呼ばれるＲＰＣ構造がある。

```
typedef struct
{
      rpc_naf_addr_alloc_fn_t naf_addr_alloc;
      rpc_naf_addr_copy_fn_t naf_addr_copy;
      rpc_naf_addr_free_fn_t naf_addr_free;
      rpc_naf_addr_set_endpoint_fn_t naf_addr_set_endpoint;
      rpc_naf_addr_inq_endpoint_fn_t naf_addr_inq_endpoint;
      rpc_naf_addr_set_netaddr_fn_t naf_addr_set_netaddr;
      rpc_naf_addr_inq_netaddr_fn_t naf_add_inq_netaddr;
      rpc_naf_addr_set_options_fn_t naf_addr_set_options;
      rpc_naf_addr_inq_options_fn_t naf_addr_inq_options;
      rpc_naf_desc_inq_addr_fn_t naf_desc_inq_addr;
      rpc_naf_desc_inq_network_fn_t naf_desc_inq_network;
      rpc_naf_inq_max-tsdu_fn_t naf_inq_max_tsdu;
      rpc_naf_get_broadcast_fn_t naf_get_broadcast;
      rpc_naf_addr_compare_fn_t naf_addr_compare;
      rpc_naf_inq_pth_unfrq_tpdu_fn_t naf_inq_max_pth_unfrq_tpdu;
      rpc_naf_inq_loc_unfrq_tpdu_fn_t naf_inq_max_loc_unfrq_tpdu;
      rpc_naf_set_pkt_nodelay_fn_t naf_set_pkt_nodelay;
      rpc_naf_is_connect_closed_fn_t naf_is_connect_closed;
      rpc_naf_twr_flrs_from_addr_fn_t naf_tower_flrs_from_addr;
      rpc_naf_twr_flrs_to_addr_fn_t naf_tower_flrs_to_addr;
      rpc_naf_desc_inq_peer_addr_fn_t naf_desc_inq_peer_addr;
  }rpc_naf_epv_t, *rpc_naf_epv_p_t;
```

【００５９】各ＮＡＦは、この構造をロードし、かつ、特定されたＮＡＦのためのrpc_g_naf_id[]テーブルのエントリにそれを付加する初期化ルーチンを持っている。この時点から、「Ｕｎｉｘネットワーク・アドレス・ファミリィ」のすべての呼び出しは、これらのＥＰＶを通※して行なわれる。「Ｕｎｉｘネットワーク・アドレス・ファミリィ」を実施するために、下記のルーチンを作成しなければならない。

【００６０】

```
0.1.2.3.1  rpc_unix_init()
PRIVATE void rpc_unix_init(naf_epv,status)
rpc_naf_epv_p_t* naf_epv;
unsigned32* status:
{
      Load the rpc_unix_epv structure with the static routines
```

defined in this file (unixnaf.c).（このファイル（unixnaf.c）中に定義されている静的ルーチンを持つrpc_unix構造をロードせよ。）
Return the NAF epv, so it can be added to the global NAF table.（大域ＮＡＦテーブルに付加できるように、NAF epvを戻せ。）
}

【００６１】

0.1.2.3.2 addr_alloc()
INTERNAL void-addr-alloc(rpc_protseq_id_naf_id, endpoint,netaddr,network options,rpc_addr,status)
rpc_addr_p_t src_rpc_addr;
rpc_addr_p_t* dst_rpc_addr;
unsigned32* status;
{
Allocate memory for destination RPC address（宛先ＲＰＣアドレスのためのメモリを割当てよ）
Copy source rpc address to destination rpc address（ソースｒｐｃアドレスを宛先ｒｐｃアドレスにコピーせよ）
}

【００６２】

0.1.2.3.3 addr_copy()
INTERNAL void addr_copy (src_rpc_addr. dist_rpc_addr. status)
rpc_addr_p_t_src_rpc_addr;
rpc_addr_p_t *dist_rpc_adr;
unsigned32 *status;
{
Allocate memory for the destination RPC address（宛先ＲＰＣアドレスのためのメモリを割り当てよ）
Copy source rpc address to destination rpc address（ソースｒｐｃアドレスを宛先ｒｐｃアドレスにコピーせよ）
}

【００６３】

0.1.2.3.4 addr_free()
INTERNAL void addr_free (rpc_addr,status)
rpc_addr_p_t* rpc_addr;
unsigned 32 *status;
{
Free memory of RPC addr and set the RPC address pointer to NULL.（ＲＰＣアドレスのメモリを開放して、ＲＰＣポインタを「無効」にセットせよ。）
}

【００６４】

0.1.2.3.5 addr_set_endpoint ()
INTERNAL void addr_set_endpoint (endpoint,rpc_add,status)
unsigned_char_p_t endpoint;
rpc_addr_p_t *rpc_addr;
unsigned32* status;
}
Check for the special case where endpoint is a pointer to

29 30

a zero length string (as opposed to NULL). In this case,
the caller is requesting that the endpoint be zeroed
out of the rpc_addr structure, so just set
rpc addr-> sa.sun_path[0] = '?0' and return.（エンドポイントが長さゼロのストリング（「無効」と対立する）へのポインタである特別の場合をチェックせよ。この場合において、rpc_addr構造からエンドポイントがゼロにされることを、呼び出し側が要求し、従ってrpc_addrをsa.sun_path[0]='?0'にセットだけで、復帰せよ。）
BEGIN AIX ONLY
Check for the existence of the RPC_UNIX_DOMAIN_DIR_PATH
environment variable.（RPC_UNIX_DOMAIN_DIR_PATH環境変数の存在をチェックせよ。）
if it exists then use it as the base for file name paths.（若し存在するならば、ファイル名パスのためのベースとしてそれを使用せよ。）
if it doesn't exist, then use the default: /var/dce/rpc/socket
on AIX.（若し存在しなければ、ＡＩＸのデフォルト/var/dce/rpc/socketを使用せよ。）
if an endpoint was specified, then check if it is
a full path(i.e. starts with '/' character) or
just a relative filename.（若しエンドポイントが特定されていたならば、それはフル・パス（即ち、'／'文字で開始する）であるか、または相対ファイル名であるかをチェックせよ。）
if it is a relative path, then append it to the base path.（若しそれが相対パスであれば、それをベース・パスに付加せよ。）
if it is an absolute path, then overwrite what has been
given as the path up to this point, no matter what it is.（若しそれが絶対パスであれば、どんなパスであれ、このポイントまでのパスとして与えられものを書き重ねよ。）
END AIX ONLY
if no endpoint was given, create one using uuid_create() and
uuid_to_string().（若しエンドポイントが与えられていなければ、uuid_create()及びuuid_to_string()を用いてエンドポイントを作成せよ。）
Now have a 32 character uuid string as the filename.（現在、ファイル名として３２文字のuuidストリングを持っている。）
append the filename to the path.（パスにファイル名を付加せよ。）
Now that a full path has been built, add the pathname to
the RPC addr structure (sa.sun_path) and set the length of
the filename.（現在、フルパスが作成されており、そのパス名をＲＰＣアドレス構造（sa.sun_path）に付加し、そしてファイル名の長さをセットせよ。）
}

【００６５】

```
0.1.2.3.6  addr_inq_endpoint()
INTERNAL void addr-inq-endpoint (rpc_addr,endpoint,status)
rpc_addr_p_t_ rpc_addr;
unsigned_char_t** endpoint;
unsigned32  *status:
{
```

Allocate memory for the filename (i.e. the endpoint).（ファイル名（即ちエンドポイント）のためのメモリを割り当てよ。）

31 32

strcpy() the filename from the RPC addr structure into the newly allocated string.（ＲＰＣアドレス構造からのファイル名を新しく割り当てられたストリング中にstrcyp()せよ。）

}

【００６６】

0.1.2.3.7 addr_set_netaddr()

INTERNAL void addr_set_netaddr(netaddr,rpc_addr,status)
unsigned_char_p_t netaddr;
rpc_addr_p_t* rpc_addr;
unsigned32* status;
{

これは不動作ルーチンである。「Ｕｎｉｘドメイン」を用いた時、netaddrの概念はない。
このルーチンを必要とする他のＮＡＦ（ネットワーク・アドレス・ファミリィ）と一貫性を持たせるために、このルーチンは存在しなければならない。

}

【００６７】

0.1.2.3.8 addr_inq_netaddr()

INTERNAL void addr_inq_netaddr(rpc_addr,netaddr,status)
rpc_addr_p_t rpc_addr;
unsigned_char_5** netaddr;
unsigned32* status;
}

これは不動作ルーチンである。「Ｕｎｉｘドメイン」を用いた時、netaddrの概念はない。
このルーチンを必要とする他のＮＡＦ（ネットワーク・アドレス・ファミリィ）と一貫性を持たせるために、このルーチンは存在しなければならない。

}

【００６８】

0.1.2.3.9 addr_set_options()

INTERNAL void addr_set_options(network_options,rpc_addr,status)
unsigned_char_p_t network_options;
rpc_addr_p_t* rpc_addr
unsigned32 *status;
{

これは不動作ルーチンである。「Ｕｎｉｘドメイン」を用いた時、netaddrの概念はない。
このルーチンを必要とする他のＮＡＦ（ネットワーク・アドレス・ファミリィ）と一貫性を持たせるために、このルーチンは存在しなければならない。

}

【００６９】

0.1.2.3.10 addr_inq_options()

INTERNAL void addr_inq_options(rpc_addr,network_options,status)
rpc_addr_p_t rpc_addr;
unsigned_char_t **network_options;
unsigned_32 *status;
{

これは不動作ルーチンである。「Ｕｎｉｘドメイン」を用いた時、netaddrの概念はない。
このルーチンを必要とする他のＮＡＦ（ネットワーク・アドレス・ファミリィ）と一貫性を持たせるために、このルーチンは存在しなければならない。）

【００７０】

```
0.1.2.3.11  inq_max_tsdu()
INTERNAL void inq_max_tsdu (naf_id, iftype,protocol,max_tsdu,status)
rpc_naf_id_t naf_id;
rpc_network_if_id_t iftype;
rpc_network_protocol_id_t protocol;
unsigned 32 *max_tsdu;
unsigned 32 *status;
{
```

これは不動作ルーチンである。パケットはネットワーク上には出ないから、ＩＰの設定は関係なし。

```
}
```

【００７１】

```
0.1.2.3.12  addr_compare()
INTERNAL boolean addr_compare (addr1, addr2, status)
rpc_addr_p_t addr1, addr2;
unsigned 32 *status;
{
    Compare the socket address file name paths of the 2 RPC
    addrs passed in.（通された２つのＲＰＣアドレスのソケット・アドレス・ファイル名パスを比較せよ。）
    Return TRUE or FALSE.（真または偽を戻せ。）
}
```

【００７２】

```
0.1.2.3.13  inq_max_pth_unfraq_tpdu()
INTERNAL void inq_max_pth_unfraq_tpdu
(rpc_addr,iftype,protocol,max_tpdu,status)
rpc_addr_p_t rpc_addr;
rpc_network_if_id_t iftype;
rpc_network_protocol_id_t protocol;
unsigned 32 *max_tpdu;
unsigned 32 *status;
```

これは不動作ルーチンである。パケットはネットワーク上には出ないから、ＩＰの設定は関係なし。

```
}
```

【００７３】

```
0.1.2.3.14  inq_max_loc_unfraq_tpdu()
INTERNAL void inq_max_loc__unfraq_tpdu
(naf_id,iftype,protocol,max_tpdu,status)
rpc_naf_id_t naf_id;
rpc_network_if_id_t iftype;
rpc_network_protocol_id_t protocol;
unsignd32* mas_tpdu;;
unsigne32* status:
{
```

これは不動作ルーチンである。パケットはネットワーク上には出ないから、ＩＰの設定は関係なし。

}

【００７４】

0.1.2.3.15 desc_inq_network()
INTERNAL void desc_inq_network(desc,socket_type,protocol_id,status)
rpc_socket_t desc;
rpc_network_if_id_t *socket_type;
rpc_network_protocol_id_t *protocol_id;
unsigned32 *status;
{
Call RPC_socket_get_if_id() to get the socket_type. (socket_typeを得るためにRPC_socket_get_if_id()を呼び出せ。)
Based on the socket type, return the protocol id. In our case it will always be rpc_c_network_protocol_id_uns (Unspecified). (ソケット・タイプに基いて、プロトコル識別子を戻せ。この実施例においては、これは、常にrpc_c_network_protocol_id_uns（不特定）である。)
}

【００７５】

0.1.2.3.16. set_pkt_nodelay()
INTERNAL void set_pkt_nodelay(desc.status)
rpc_socket_t desc;
unsigned32* status;
{
これは不動作ルーチンである。パケットはネットワーク上には出ないから、ＩＰの設定は関係なし。
}

【００７６】

0.1.2.3.17 is_connect_closed()
INTERNAL boolean is_connect_closed(desc,status)
rpc_socket_t desc;
unsigned 32 *status;
{
このルーチンは常に真を返却する。
}

【００７７】

0.1.2.3.18 tower_flrs_from_addr()
INTERNAL void tower_flrs_from_addr (rpc_addr,lower_flrs,status)
rpc_addr_p_t rpc_addr;
twr_p_t *lower_flrs;
unsinged 32 *status;
{
Get the network protocol id (aka transport layer protocol) for this RPC addr. (このＲＰＣアドレスのためのネットワーク・プロトコル識別子を獲得せよ（akaトランスポート層プロトコル）。)
Use the network protocol id as a parameter to the routine twr_unix_lower_flrs_from_sa(). (ルーチンtwr_unix_lower_flrs_from_sa()に対するパラメータとしてネットワーク・プロトコル識別子を使用せよ。)

37 38

このルーチンは、「Ｕｎｉｘドメイン」ＲＰＣアドレスのタワー参照子の下位フロアを示すタワーのオクテットを戻す。twr_unix_lower_flrs_from_sa()ルーチンの細部に関してはファイルtwr_unix.c（本明細書で後述する）を参照されたい。

}

【００７８】

```
0.1.2.3.19  tower_flrs_to_addr()
INTERNAL void tower_flrs_to_addr (tower_octet_string,
rpc_addr, status)
byte_p_t tower_octet_string;
rpc_addr_p_T *rpc_addr;
unsigned 32 *status;
{
```

Convert the lower floors of a tower to a sockaddr, calling the routine twr_unix_lower_flrs_to_sa().（ルーチンtwr_unix_lower_flrs_to_sa()を呼び出して、タワーの下位フロアをソケット・アドレスに変換せよ。）

twr_unix_lower_flrs_from_sa()ルーチンの細部に関してはファイルtwr_unix.c（本明細書で後述する）を参照されたい。

Add the socket address to an RPC address and return it.（ソケット・アドレスをＲＰＣアドレスに加え、それを戻せ。）

```
}
```

【００７９】

```
0.1.2.3.20  desc_inq_peer_addr()
INTERNAL void desc_inq_peer_addr (protseq_id,desc,rpc_addr,status)
rpc_protseq_id_t_ protesq_id
rpc_socket_t desc;
rpc_addr_p_t *rpc_addr;
unsigned32 *status;
}
```

Allocate memory for the new RPC address, and fill in the protseq id and length of the sockaddr structure.（新しいＲＰＣアドレス用のメモリを割り当て、そしてprotseq識別子及びソケット・アドレス構造の長さを満たせ。）

Call rpc_socket_inq_endpoint(), which will fill in the peer endpoint, which is always the same as the current processes endpoint, in the case of Unix Domain.（「Ｕｎｉｘドメイン」の場合、現在のプロセス・エンドポイントと常に同じであるピア・エンドポイント中に満たすrpc_socket_inq_endpoint()を呼び出せ。）

```
}
```

【００８０】0.1.2.4 src/rpc/runtime/unixnaf.h
このファイルは、作成しなければならない新しいファイルである。このファイルは、主として、「ＵｎｉｘのＮＡＦ epv」ルーチンのためのプロトタイプを含んでいる。加えて、このファイルは、「ｕｎｉｘドメイン」ＲＰＣアドレスの表示の定義を含んでいる。これは以下のように定義される。

```
typedef struct rpc_addr_unix_t
{
    rpc_protseq_id_t rpc_protseq_id;
    unsigned32 len;
    struct sockaddr_un sa;
} rpc_unix_addr_t, *rpc_unix_addr_p_t;
```

【００８１】ＲＰＣがコマンド・コードを実行している時に、ＲＰＣアドレスは、疑似透明構造（rpc_addr_t）として通過されるが、しかし、この構造がＮＡＦ特定ルーチンに通された時、この構造は使用するファミリィ用

のＲＰＣアドレス構造に与えられる。ｕｎｉｘストリームの場合、この構造はrpc_unix_addr_tに与えられる。

【００８２】このファイルに対して必要とする他の付加部分は、ＡＩＸオペレーティング・システムにおける「Ｕｎｉｘドメイン」ソケット呼び出しに使用するためのファイル名を作成するために使用されるデフォルト・パス名である。これは、「オペレーティング・システ＊ム」特定のものであり、ＯＳ／２に関しては必要としない。エンドポイントを作成する時に、若しエンドポイントが存在しないか、または使用されるエンドポイントがフル・パス名を特定しなければ、デフォルト・パス名が用いられる。デフォルト・パス名は下記のようにして定義される。

```
#ifdefined(AIX_RROD)
#define RPC_DEFAULT_UNIX_DOMAIN_PATH "_" /var/dce/rpc/socket"
#endif
```

【００８３】ＯＳ／２オペレーティング・システムに関しては、「Ｕｎｉｘドメイン」ソケット・ファイルは内部的のみに使用される。ユーザが目視できるファイルは作成されない。ＯＳ／２オペレーティング・システムは、ソケット・ファイルに関連した管理的なタスク（即ち、整理動作）を取り扱う。特別なパスを生成する必要はない。

【００８４】0.1.2.5 src/rpc/runtime/RIOS/unixnaf_sys.c

これは、作成しなければならない新しいファイルである。このファイルは、「Ｕｎｉｘドメイン・ネットワー※ク・アドレス・ファミリィ」に付属するような、ＲＩＯＳプラットフォームに対してシステム特定のルーチンを含んでいる。

【００８５】0.1.2.5.1 rpc_unix_desc_inq_addr()

ファミリィ、エンドポイント及びネットワーク・アドレスを得るために問い合せられたソケット記述子を受信する。若しこの情報が「Ｕｎｉｘドメイン」アドレスに対して有効ならば、ソケットから得られた情報によって初期化されるＲＰＣアドレス用のスペースが割り当てられる。作成されたＲＰＣアドレスを表示したアドレスはrpc_addr中に戻される。

```
PRIVATE void rpc_unix_desc_inq_addr(protseq_id,desc,rpc_addr_vec,statu
s)
  rpc_protseq_id_t protseq_id;
  rpc_socket_t desc;
  rpc_addr_vector_p_t *rpc_addr_vec;
  unsigned 32 *status;
  {
        Simply do a "getsockname" into a local Unix Domain RPC
        address.（ローカル「Ｕｎｉｘドメイン」ＲＰＣアドレスの中にソケット名を単純に獲得せよ。）
        Allocate memory for the RPC address, and RPC address vector.（ＲＰＣアドレス及びＲＰＣアドレス・ベクトル用のメモリを割り当てよ。）
        Fill in the RPC address structure with the protocol sequence
        id, and the Unix sockaddr structure.（ＲＰＣアドレス構造の中に、プロトコル順序識別子及びＵｎｉｘソケット・アドレス構造を満たせ。）
        Add the RPC address to the RPC address vector.（ＲＰＣアドレスをＲＰＣアドレス・ベクトルに加えよ。）
  }
```

【００８６】0.1.2.5.2 rpc_unix_get_broadcast()

このルーチンは不動作であるけれども、必要なルーチン★

```
PRIVATE void rpc_unix_get_broadcast(naf_id,protseq_id,rpc_addr_vec,sta
tus)
  rpc_naf_id_t naf_id;
  rpc_protseq_id_t protseq_id;
  rpc_addr_vector_p_t *rpc_addr_vec;
  unsigned32 *status;
  {
        Do nothing. Set output parameters to NULL and return.（何もせず。出力パラメータを「無効」にセットし、復帰せよ。）
```


★であり、NAF_epvの中にエントリがある。

}

【００８７】0.1.2.6　src/rpc/runtime/twr_unix.h
これは新しいファイルであって、作成しなければならない。このファイルは宣言と、twr_unix.cファイルに特定するプロトタイプとを含んでいる。現在、このファイルだけが、ルーチンrpc_rpcd_is_running()のためのプロトタイプを含んでいる。

【００８８】0.1.2.7　src/rpc/runtime/twr_unix.c
これは新しいファイルであって、作成しなければならない。このファイルは、タワー表示からソケット・アドレスへの変換を行なうことと、ソケット・アドレスからタワー表示への変換を行なうこととを特定するルーチンを含んでいる。また、このファイルは、ＲＰＣランタイム及びＲＰＣＤの両方によって使用されるユティリティ・ルーチンを含んでいる。

【００８９】0.1.2.7.1　rpc_rpcd_is_running()
これは新しいルーチンである。このルーチンは、下記の２つの目的のために使用される。２つの目的とは（１）ＲＰＣＤのインスタンスが既に実行されているか否かを＊決定するための初期化の間で、ＲＰＣＤがこのルーチンを使用する目的と、（２）ローカル・ホスト・コンピュータ中のＲＰＣＤが実行されているか否かと、このプロシージャがncacn_unix_streamプロトコル順序をサポートするか否かとを決定するためにＲＰＣランタイムがこのルーチンを使用する目的とである。

【００９０】このルーチンは、ucacn_unix_streamをサポートするＲＰＣＤのインスタンスをチェックする。これは、ＲＰＣＤエンドポイント・ファイルの存在を検索することによって行なわれる。若しこのファイルが存在しなければ、ＲＰＣＤは、Ｕｎｉｘストリームのサポートによって実行されない。若しこのファイルが存在すれば、このファイルは、ＲＰＣＤインスタンスへの接続を試みるために使用される。若し接続が不成功に終わったならば、ＲＰＣＤはＵｎｉｘストリームのサポートによって実行されない。このファイルは除去される。若し接続が成功したならば、ＲＰＣＤは実行される。このファイルは除去されない。

```
PRIVATE boolean32 rpc_rpcd_is_running(status)
unsigned32 *status;
{
        Check if the Unix Stream protocol sequence is supported.
        If not, return the error rpc_s_protseq_not_supported. (Ｕｎｉｘストリーム・プロトコル順序がサポートされているか否かをチェックせよ。若しサポートされていなければ、エラーrpc_s_protseq_not_supportedを戻せ。)
        Inspect the instance specification for the RPCD(ept_
        v30_cifspec) and get the well-known endpoint for the
        Unix Stream protocol sequence. (ＲＰＣＤ（ept_v30_cifspec）のインスタンス仕様を検査し、そしてＵｎｉｘストリーム・プロトコル順序の公知のエンドポイントを獲得せよ。)
        The endpoint is used as a socket file for communicating to
        the RPCD. First, check if the file exists. If it does exist,
        then we expect it to be a file associated with a socket, and
        the fopen() should fail with the appropriate error. If it
        doesn't exist, then there is no rpcd running (at least with
        support for ncacn_unix_stream). So just return. Everything
        is OK. (そのエンドポイントはＲＰＣＤと通信するためのソケット・ファイルとして使用される。先ず、そのファイルが存在するか否かをチェックせよ。若しそのファイルが存在すれば、そのファイルはソケットに関連されたファイルであると想定され、そしてfopen()は適当なエラーで不成功にされる。若しそのファイルが存在しなければ、ＲＰＣＤ（少なくともncacn_unix_streamをサポートするＲＰＣＤ）は実行されない。従って、戻るだけである。すべてＯＫである。)
        If this file can be opened, then it is NOT a socket file.
        Therefore remove it (i.e. it shouldn't be therein the
        first place) and return. (若しこのファイルがオープンできれば、それはソケット・ファイルではない。従って、それを除去し（即ち、最初の位置ではない）、そして戻れ。)
        If the file exists, and is a socket file, check if there is
```

an RPCD actively using it. Do this via a socket()/connect()
sequence.（若しこのファイルが存在し、ソケット・ファイルであれば、それを積極的に使用するＲＰＣＤがあるか否かをチェックせよ。これを、socket()/connect()順序を介して行なえ。）
If no rpcd is running, we expect the connect call to fail.
We check for the proper error, and flag the case where an
unexpected error occurs.（若しＲＰＣＤが実行されていなければ、接続呼び出しは不成功に終わると予測される。ユーザはエラーの適正をチェックし、そして、予想外のエラーが発生した場合、フラグする。）
If able to connect to the RPCD, return true. Otherwise,
return false.（若しＲＰＣＤへの接続が成功したならば、真を戻せ。ＲＰＣＤへの接続が不成功ならば、偽を戻せ。）
}

【００９１】0.1.2.8 src/rpc/runtime/twrp.h
これは既存のファイルである。このファイルは、ＲＰＣタワー参照子構造及びオクテット・ストリングの構造及び検査に関する宣言を含んでいる。このファイルは下記の付加的なラインを含ませるために変更される。

```
/* Protocol identifiers for each lower tower floor. */
#define twr_c_flr_prot_id_fname 0x20 /* Unix Domain filename */
#define twr_c_flr_prot_id_dummy 0x21 /* Dummy prot id for floor 5 */
/* Number of lower floors in Unix address family. */
#define twr_c_num_unix_lower_flrs 2 /* Number lower flrs in unix tower */
/* Unix family filename size */
#define twr_c_unix_frame_size 108
```

【００９２】注記 ：「Ｕｎｉｘドメイン」タワー参照子はタワー・フロアを４個しか使用しないけれども、しかし、第５のフロアが含まれており、これはデータを含んでいない。これは、５個のフロアが「フル・タワー」であると見做されるタワー・オクテット・ストリングを検査するのに用いられるＲＰＣランタイムのアルゴリズムに原因がある。「Ｕｎｉｘドメイン」はフル・タワーであるけれども、タワーに関連するデータを持つフロアは４個だけしかない。

【００９３】0.1.2.9 src/rpc/runtime/comtwrref.c
これは既存のファイルである。このファイルはプロトコル・タワーのランタイム参照子表示に関して動作するルーチンを含んでいる。

【００９４】0.1.2.9.1 rpc_tower_ref_inq_protseq_id()
このルーチンは変更される。このルーチンは、そのプロトコル順序のためのタワー参照子を作る上位フロア３及び下位タワー・フロアのためのプロトコル識別子の組み合せに対して使用可能なすべてのプロトコル順序のマッピングを含む静的テーブルを含んでいる。このテーブルは、ncacn_unix_streamプロトコル順序用のエントリの付加を含ませるために変更されなければならない。

```
static_rpc_tower_prot_ids_t rpc_tower_prot_ids[rpc_c_protseq_id_max + 1] =
{
    {rpc_c_protseq_id__ncacn_unix_stream,3,
        {{rpc_c_cn_proto_id,{0}},
            {twr_c_flr_prot_id_fname,{0}},
            {twr_c_flr_prot_id_dummy,{0}},
            {0x00,{0}}
        }
    },
    .../* other existing table elements */
};
```

【００９５】0.1.2.10 src/rpc/rpcd/rpcd.c
これは既存のファイルである。このファイルは、rpcdサーバが来た時、ncacn_unix_streamを介する要求を聴取するのに使用されるソケット・ファイルが存在するか否かを、rpcdサーバがチェックするように、変更される必要がある。若しソケット・ファイルが存在していれば、

そのソケット・ファイルは削除する必要がある。これは、ファイルが既に存在しているならば、ソケット・ファイルは作成できないからである。

【００９６】0.1.2.10.1 main() *

```
....* database initialization */
....
if (rpc_rpcd_is_running(&status))
{
      fprintf(stderr,
      "(rpcd) Instance already running. Can't continue.?n");
      exit(1);
}
if((status ! = rpc_s_ok) && (status ! = rpc_s_protseq_not_supported))
}
      fprintf(stderr,
      "(rpcd) Error checking for other rpcd instances.?n");
      exit(1);
}
....
....* rest of rpcd setup */
```

＊データベースが初期化された後で、プロトコル順序のサポートが設定される前に、下記のコード付加されなければならない。

【００９７】0.1.2.11 src/libdce/RIOS/libdce.syms

これは既存のファイルである。このファイルは、ルーチンrpc_rpcd_is_running()を含ませるために更新する必要がある。これは、libdce.aの外側のルーチンまたはプログラムがこのルーチンにアクセスできるようにするためである。rpcdがこのルーチンを呼び出すので、このルーチンが必要である。下記のラインをlibdce.-symsに付加するだけでよい。

rpc_rpcd_is_running

【００９８】0.1.2.12 Auto Handles

[auto_handle]の属性の使用は、アプリケーション・コードがバインディング・ハンドルを獲得し及び／又はそのバインディング・ハンドルのエンドポイントを分析するための責任を持たないことを表示する。[auto_handle]の属性は、ＲＰＣランタイムがこれのすべてを操作することを特定する。ユーザが特定するすべてのものは、ＲＰＣランタイムがバインディング・ハンドルの検索を開始するネーム・スペースのエントリ位置を特定するだけである。（これは、環境変数RPC_DEFAULT_ENTRYを介して行なわれる。）クライアント及びサーバが同じホス※ト・コンピュータに存在する場合に、ＲＰＣランタイムは、ncacn_unix_streamを使用するインテリジェンスを持つ。

【００９９】0.1.2.13 Utilities

この機能の実施は管理的な責任を遂行するための幾つかのユティリティの作成を必要とする。これらのユティリティは下記に説明されている。

【０１００】0.1.2.13.1 src/rpc/utils/rpcclean

このユティリティは、ＲＰＣランタイムのルーチンrpc_network_inq_protseqs()を単に呼び出し、そして、呼び出されたプロトコル順序のストリング・ベクトルを、１行に１プロトコル順序づつ印刷する。このユティリティは、与えられたホスト・コンピュータに使用可能なプロトコル順序を決定するためのmkdceに対して使用される。１実施例において、サポートされたプロトコル順序はmkdce記述子に配線される。また、この記述子は、与えられたホスト・コンピュータにおいてサポートされたプロトコル順序を決定するための管理用の迅速な方法に使用することができる。このユティリティのためのコードは以下の通りである。

```
*include <stdio.h>
*include <dce/rpc.h>
*include <dce/ecd_error.h>
char message[dce_c_error_string_len];
main(unsigned32argc,unsigned_char_p_t argv[])
{
      rpc_protseq_vector_p_t psvp;
      unsigned32i,status=0,tmp_status=);
      rpc_network_inq_protseqs( &psvp, &status);
      if(status ! = rpc_s_ok)
```

47 48

```
{
        dce_error_inq_text(status,message,&tmp_status);
        printf("%s: %s/n, argv[0], message);
        exist(1);
}
for (i = 0;i<psvp->count;i+ +)
{
        printf("%s/n",psvp->protseq[i]);
}
}
```

【０１０１】0.1.2.13.2 src/rpc/utils/rpcprotseqs このユティリティはＲＰＣランタイムのルーチンrpc_network_inq_protseqs()を単純に呼び出し、そして、呼び出されたプロトコル順序のストリング・ベクトルを、１行に１プロトコル順序づつ印刷する。このユティリティは、与えられたホスト・コンピュータに使用可能なプロトコル順序を決定するためのmkdceに対して使用される。１実施例において、サポートされたプロトコル順序はmkdce記述子に配線される。また、この記述子は、与えられたホスト・コンピュータにおいてサポートされたプロトコル順序を決定するための管理用の迅速な方法に使用することができる。このユティリティのためのコードは以下の通りである。

```
#include <stdio.h>
#include <dce/rpc.h>
#include <dce/dce_error.h>
char message[dce_c_error_string_len];
main(unsigned32 argc,unsigned_char_p_t argv[])
{
        rpc_protseq_vector_p_t psvp;
        unsigned32 i,status=0,tmp_status=0;
        rpc_network_inq_protseqs(&psvp,&status);
        if(status ! = rpc_s_ok)
        {
                dce_error_inq_text(status,message,&tmp_status);
                printf("%s; %s/n, argv[0], message);
                exit(1);
        }
        for (i = 0;i<psvp->count;i+ +)
        {
                printf("%s/n",psvp->protseq[i]);
        }
}
```

【０１０２】0.1.3 互換性
この機能は互換性の問題を全く生じない。
【０１０３】0.1.3.1 逆方向の互換性
既存のアプリケーションはこの機能を含ませるためにＤＣＥをアップグレードすることができ、修正を施すことなく実行することができる。また、クライアント／サーバ・アプリケーションが異なったホスト・コンピュータにあり、一方のアプリケーションがこの機能を使用し、他方のアプリケーションがこの機能を使用しない場合でも、修正することなく実行することができる。複数のコンピュータに跨がって「Ｕｎｉｘドメイン」を使用することはできないので、上述のことは、互換性の問題ではなく、共存の問題である。

【０１０４】0.1.3.2 Cross-Platformの互換性
この機能は、ＡＩＸ及びＯＳ／２を使用して実行される。ＡＩＸ及びＯＳ／２プラットフォームの間を通信するクライアント／サーバ・アプリケーションは、この機能の利益を享受することができないけれども、既存のアプリケーションは、この機能を含ませるためにＤＣＥをアップグレードすることができ、そして修正することなく実行することができる。
【０１０５】逆方向の互換性は、この機能の導入前に可能であった範囲内で、プラットフォームに跨がってのみ拡大する。この機能の範囲を越えたソースからの他の非互換性の導入があるにも拘らず、勿論、互換性のすべてのクレームがある。

49

【０１０６】0.1.4　ＤＣＥの導入及び構成の影響
ＤＣＥがコンピュータに導入される時、ディレクトリ/var/dce/rpc/socketが作成されなければならない。若しこのディレクトリが既に存在しているならば、そのディレクトリは消去され、再作成されねばならない。これは、ＤＣＥ導入のアップグレードが起きた時、すべてのＤＣＥアプリケーションが停止されるものと仮定され、「Ｕｎｉｘドメイン」ソケット・ファイルが作成される場合、このディレクトリ中に残されたすべてのファイルは古いファイルとなるからである。

【０１０７】0.1.5　ＲＡＳ（信頼性、可用性、保守性）に対する影響
この機能によってＲＡＳに及ぼす影響はない。

【０１０８】0.1.6　ＮＬＳ（各国語サポート）に対する影響
この機能によってＮＬＳに及ぼす影響はない。

【０１０９】0.1.7　性能及び記憶域の見積り
ＲＰＣの性能は、クライアント及びサーバが同じホスト・コンピュータ中に存在する場合、１０％乃至２０％までの性能向上がある。クライアント及びサーバが異なったホスト・コンピュータ中にある場合でも、性能は影響されない。

【０１１０】記憶域の見積りは問題にはならない。サーバが登録する各「Ｕｎｉｘドメイン」のエンドポイントに対してソケット・ファイルが作成されるけれども、このファイルは長さが０である。

【０１１１】システム中に取り残されている古いソケット・ファイルを除去するために、rpccleanユティリティ（この明細書において既に説明されている）を実行することができる。これは、ファイルシステム中に用いられているｉノードの数を減小する。

【０１１２】0.1.8　テストの計画
テストの計画はこの明細書には記載しない。この機能が正しく動作することを保証するのに必要とする付加的なテストケースまたはテスト構成を示した現用のＲＰＣＦＶＴテスト計画の付属書を参照されたい。

【０１１３】本発明の良好な実施例は、パーソナル・コンピュータ、またはワークステーションのランダム・アクセス・メモリ中に常駐するコード・モジュール中のインストラクションの組として実行される。コンピュータ・システムによって要求されるまで、インストラクションの組は、例えばハード・ディスク・ドライブ、またはオプティカル・ディスク（ＣＤ－ＲＯＭ中に使用されている）や、フロッピィ・ディスク（フロッピィ・デイスク・ドライブ中で使用されている）などの取り外し可能なメモリなどの他のコンピュータ・メモリ中にストアすることができる。

【０１１４】以上、本発明は特定のオペレーティング・システム及びネットワーク環境における良好な実施例について説明してきたが、本発明の技術思想の範囲内で、

50

本発明の実施例を他のオペレーティング・システム及びネットワーク・アーキテクチャによって変更し、または置換することは当業者であれば容易に行なうことができるのは自明であろう。従って、例えば、ＤＣＥ－ＲＰＣアーキテクチャの術語において、クライアント・プロセスはサーバ・プロセスに「バインディング・ハンドル」を渡す。然しながら、本発明はＤＣＥ－ＲＰＣアーキテクチャに限定して解釈されるべきではなく、従って、本発明は、例えば、サーバ・プロセスの位置を特定し、そしてクライアント及びサーバ・プロセスの間で使用される通信プロトコルを設定するデータ構造を、サーバ・プロセスに通過することなどにより、クライアント・プロセスがローカル通信を獲得し、かつ設定する任意のネットワーク環境をカバーするようなより包括的な観点から解釈されるべきである。

【０１１５】まとめとして、本発明の構成に関して以下の事項を開示する。

【０１１６】（１）分散計算環境内におけるクライアント・プロセスとサーバ・プロセスの間の通信を管理する方法であって、上記クライアント・プロセスは、トランスポート層及びネットワーク層を有する物理ネットワークに接続されたホスト・コンピュータ中に存在し、上記方法は、（ａ）　クライアント・プロセスによって行なわれた遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中に存在するか否かを決定するステップを含み、上記ＲＰＣは、トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を持ち、（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中に存在するならば、上記ＲＰＣのプロトコル順序には関係なく、上記クライアント・プロセスと上記サーバ・プロセスとの間にインタープロセス通信路を設定するステップと、（ｃ）　上記ＲＰＣのプロトコル順序を上記クライアント・プロセスに戻すステップと、（ｄ）　上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、を含む通信管理方法。

（２）上記プロトコル順序はＲＰＣに影響を与えないで使用されることを特徴とする（１）に記載の通信管理方法。

（３）上記プロトコル順序は接続指向プロトコル順序であることを特徴とする（１）に記載の通信管理方法。

（４）上記プロトコル順序は非接続プロトコル順序であることを特徴とする（１）に記載の通信の管理方法。

（５）上記遠隔プロシージャ呼び出しは、上記ホスト・コンピュータのオペレーティング・システムのメッセージ送受信機能を用いて実行されることを特徴とする（１）に記載の通信管理方法。

（６）分散計算環境内におけるクライアント・プロセスとサーバ・プロセスの間の通信を管理する方法であっ

て、上記クライアント・プロセスは、トランスポート層及びネットワーク層を有する物理ネットワークに接続されたホスト・コンピュータ中に存在し、上記方法は、

（ａ）　遠隔プロシージャ呼び出し（ＲＰＣ）が上記クライアント・プロセスによって行なわれた時、上記遠隔プロシージャ呼び出しによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中に存在しているか否かを決定するステップと、（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中にあれば、トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を含む第１データ構造を上記クライアント・プロセスに戻すステップと、（ｃ）　上記第１データ構造を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を定義するプロトコル順序を含む第２データ構造にマップするステップと、（ｄ）　上記第２データ構造中のプロトコル順序によって定義された上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、を含む通信管理方法。

（７）上記遠隔プロシージャ呼び出しは、上記ホスト・コンピュータのオペレーティング・システムのメッセージ送受信機能を用いて実行されることを特徴とする（６）に記載の通信管理方法。

（８）上記第１データ構造のプロトコル順序は上記ＲＰＣに影響を与えることなく使用されることを特徴とする（６）に記載の通信管理方法。

（９）上記第１データ構造のプロトコル順序は接続指向プロトコル順序であることを特徴とする（６）に記載の通信管理方法。

（１０）上記第２データ構造のプロトコル順序はソケット・ファイルに対するフル・パス名を含んでいることを特徴とする（９）に記載の通信管理方法。

（１１）命名規則及びエンドポイントが上記パス名を決定するのに用いられることを特徴とする（１０）に記載の通信管理方法。

（１２）ホスト・コンピュータがＵＮＩＸベースのオペレーティング・システムをサポートしており、クライアント・プロセスがトランスポート層及びネットワーク層を有する物理ネットワークに接続された上記ホスト・コンピュータ中に常駐している分散計算環境内において、上記クライアント・プロセスが遠隔プロシージャ呼び出しを行なった時に、上記クライアント・プロセス及びサーバ・プロセス間の通信を管理する方法であって、

（ａ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中に存在するならば、上記遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を持つバインディング・ハンドルを上記クライアント・プロセスに戻すステップと、（ｂ）　上記遠隔プロシージャ呼び出しに通常関連しているプロトコル順序を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を設定する代替プロトコル順序にマップするステップと、（ｃ）　上記ＵＮＩＸベースのオペレーティング・システムのメッセージ送受信機能を用いて、上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、を含む通信管理方法。

（１３）「ncacn」は接続指向ＲＰＣプロトコルであり、「unix」は「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）通信ドメイン」を識別し、「stream」はＵＮＩＸドメイン・ソケットを識別するものとした場合、上記遠隔プロシージャ呼び出しに関連したプロトコル順序は接続指向プロトコル「ncacn_ip_tcp」であり、上記代替プロトコル順序は「ncacn_unix_stream」であることを特徴とする（１２）に記載の通信管理方法。

（１４）「ncadg」は非接続ＲＰＣプロトコルであり、「unix」は「ＵＮＩＸネットワーク・アドレス・ファミリィ（ＡＦ＿ＵＮＩＸ）通信ドメイン」を識別し、「dgram」はＵＮＩＸドメイン・ソケットを識別するものとした場合、上記遠隔プロシージャ呼び出しに関連したプロトコル順序は非接続指向プロトコル「ncacn_ip_udp」であり、上記代替プロトコル順序は「ncadg_unix_dgram」であることを特徴とする（１２）に記載の通信管理方法。

（１５）ホスト・コンピュータがインタープロセス通信（ＩＰＣ）機構を有し、かつ、ユーザが分散されたリソース及びプロセス・アプリケーションにアクセスすることのできる分散計算環境を与えるローカル・エリア・ネットワークであって、クライアント・プロセスからの遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出する検出手段と、該検出手段に応答して、予期されたプロトコル順序を上記クライアント・プロセスに戻し、かつ、上記ＲＰＣを助長するために代替プロトコル順序を用いる手段を含み、上記代替プロトコル順序はＩＰＣ構造を通る通信路を設定することと、を含むローカル・エリア・ネットワーク。

（１６）ユーザが分散されたリソース及びプロセス・アプリケーションにアクセスすることのできる分散計算環境を与え、かつ、トランスポート層及びネットワーク層を有するローカル・エリア・ネットワークに接続されたホスト・コンピュータを含むコンピュータ・システムであって、クライアント・プロセスからの遠隔プロシージャ呼び出し（ＲＰＣ）に応答して、該ＲＰＣによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出する検出手段と、該検出手段に応答して、予期されたプロトコル順序を上記クライアント・プロセスに戻し、かつ、上記ＲＰＣを助長するために代替プロトコル順序を用いる手段を含み、上記代替プロ

トコル順序の使用は上記クライアント・プロセスに影響を与えないことと、を含むコンピュータ・システム。

（１７）ホスト・コンピュータによって読み取り可能であり、かつ、上記ホスト・コンピュータで実行されるクライアント・プロセスからの通信を管理する方法を実施するための、該ホスト・コンピュータによって実行可能な命令からなるプログラムを具現化するプログラム記憶装置であって、上記ホスト・コンピュータはトランスポート層及びネットワーク層を有するローカル・エリア・ネットワークに接続されており、上記方法は、（ａ）　遠隔プロシージャ呼び出しが上記クライアント・プロセスによって行なわれた時、上記遠隔プロシージャ呼び出しによって識別されたサーバ・プロセスが上記ホスト・コンピュータ中にあるか否かを検出するステップと、（ｂ）　若し上記サーバ・プロセスが上記ホスト・コンピュータ中にあるならば、上記トランスポート層及びネットワーク層の使用を介した通信路を定義するプロトコル順序を含む第１データ構造を上記クライアント・プロセスに戻すステップと、（ｃ）　上記第１データ構造を、上記クライアント・プロセス及び上記サーバ・プロセス間のインタープロセス通信路を定義するプロトコル順序を含む第２データ構造にマップするステップと、　＊（ｄ）　上記第２データ構造のプロトコル順序によって定義された上記インタープロセス通信路を介して上記遠隔プロシージャ呼び出しを実行するステップと、を含むプログラム記憶装置。

【図面の簡単な説明】

【図１】本発明が適用されるコンピュータ・ネットワークを説明するための図である。

【図２】物理的ネットワークのネットワーク層及びトランスポート層を使用した従来のＲＰＣを説明するための図である。

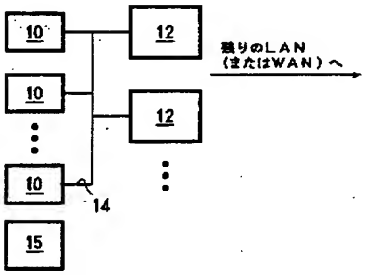
【図３】ローカルＲＰＣがホスト・コンピュータのＩＰＣ構造を用いて遂行された本発明の実施例を説明するための図である。

【図４】ローカル遠隔プロシージャ呼び出しを与えるためのプロシージャを最適化するための本発明の実施例を説明するための図である。

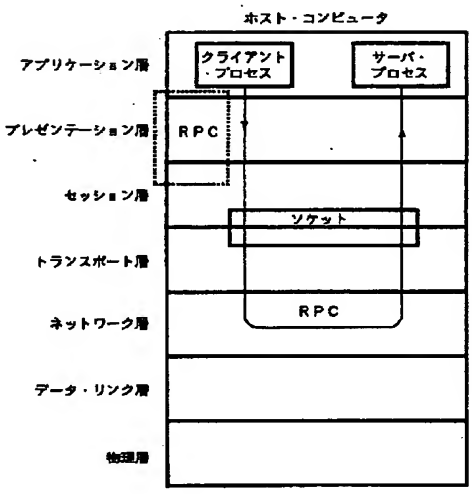
【符号の説明】

１０　クライアント
１２　サーバ
１４　ネットワーク
１５　クライアント／サーバ

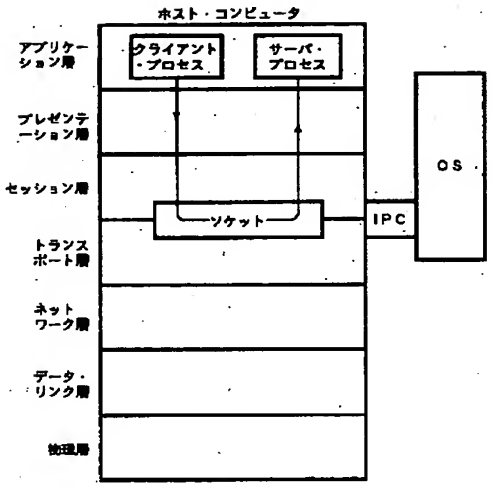
【図１】



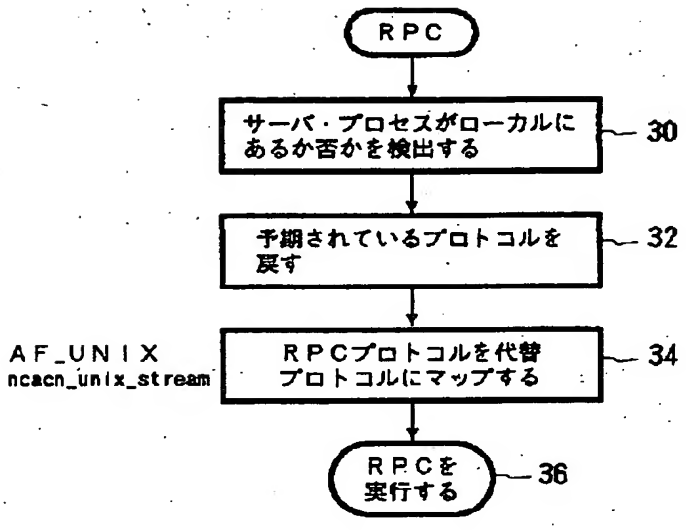
【図２】



【図３】



【図４】



フロントページの続き

(72)発明者　サンダヤ・カプール
アメリカ合衆国　78759、テキサス州、オースチン、バーカー・リッジ・ドライブ　5711

(72)発明者　イ・シュー・ウェイ
アメリカ合衆国　78726、テキサス州、オースチン、ベックウッド・ドライブ　10611